Mieke K. Malmberg
(SBN 209992)
SKIERMONT DERBY LLP
800 Wilshire Blvd., Ste. 1450
Los Angeles, CA 90017
Phone: (213) 788-4500
Fax: (213)788-4545
mmalmberg@skiermontderby.com

Paul J. Skiermont (*pro hac vice*)
Sadaf R. Abdullah (*pro hac vice*)
SKIERMONT DERBY LLP
1601 Elm St., Ste. 4400
Dallas, TX 75201
Phone: (214) 978-6600
Fax: (214) 978-6601
pskiermont@skiermontderby.com
sabdullah@skiermontderby.com
(Additional counsel identified on signature page)

Attorneys for Plaintiff
BELL NORTHERN RESEARCH, LLC

IN THE UNITED STATES DISTRICT COURT
SOUTHERN DISTRICT OF CALIFORNIA

BELL NORTHERN RESEARCH, LLC, Plaintiff, v. HUAWEI DEVICE (DONGGUAN) CO., LTD, HUAWEI DEVICE (SHENZHEN) CO., LTD., and HUAWEI DEVICE USA, INC., Defendants.	C.A. No. 3:18-CV-01784-CAB-BLM BELL NORTHERN RESEARCH, LLC'S PATENT L.R. 3.1 DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS

Pursuant to Patent L.R. 3.1 and the Court's Case Management Order, Plaintiff Bell Northern Research, LLC ("BNR") hereby provides its Disclosure of Asserted Claims and Infringement Contentions, including the claim charts attached as Exhibits A-H and all Annexes thereto. This disclosure is made solely for the purpose of the above-captioned action, with respect to above-named Defendants.

Discovery in this matter is at an early stage and BNR's investigation of Defendants' infringement is ongoing. For example, Defendants have not yet produced any documents relating to the accused products; nor have they provided any deposition testimony in this action. This disclosure is therefore based upon information that BNR has been able to obtain publicly, together with BNR's current good faith beliefs regarding the Accused Instrumentalities in this matter. For example, the figures shown in the accompanying claim charts, including but not limited to those depicting the location of and identifying the presence of claim elements in the Accused Instrumentalities, are illustrative and may change after production and review of Defendants' confidential information. Accordingly, BNR explicitly reserves the right to amend and/or supplement the accompanying claim charts regarding direct and/or indirect infringement, as well as literal infringement and/or infringement under the doctrine of equivalents based on evidence uncovered in this litigation.

The claim charts attached as exhibits hereto are not designed to represent the entire scope of infringement of the Accused Instrumentalities, but are merely to illustrate examples of how the Accused Instrumentalities infringe the Asserted Claims of the Patents-in-Suit. Further, the division of claim elements in the charts attached as exhibits hereto is provided solely for purposes of convenience in presenting BNR's Infringement Contentions. The division is not meant to modify the claim language or to inform claim construction.

The following disclosures should not be construed as BNR's preliminary construction of any asserted claim or claim term. This disclosure does not represent BNR's position on whether any claim term should be or needs to be construed,

clarified, or interpreted by the Court. Further, these disclosures also are based at least in part on BNR's present understanding of the meaning and scope of the asserted claims of the patents-in-suit in the absence of claim construction proceedings for all the patents-in-suit or substantial discovery in this matter. BNR reserves the right to supplement or amend these disclosures if its understanding of the claim terms changes, including if the Court construes them.

This disclosure is given without prejudice to BNR's rights, and BNR hereby expressly reserves its rights under Patent L.R. 3.6(a) or any other applicable basis to further supplement or amend its contentions, including without limitation, to add Asserted Claims or Accused Instrumentalities as additional facts are ascertained, analyses are made, research is completed, contentions are made, claims are construed, and Defendants' confidential information is received and reviewed in discovery.

a. Asserted Claims (Patent L.R. 3.1(a)):

Based on information presently available, BNR asserts that each of the following claims is infringed by Defendants. In accordance with the Court's instructions at the October 22, 2018 Case Management Conference, BNR has limited its assertions to no more than seven claims per patent. BNR expressly reserves the right to modify, substitute, change, or amend which claims it asserts against Defendants as more information becomes available, including without limitation the parties' claim construction positions and the Court's claim construction rulings.

U.S. Patent No.	Asserted Claims
7,319,889 ("'889 Patent")	1, 2, 4, 5, 6, 8, 12
8,204,554 ("'554 Patent")	1, 2, 4, 5, 7, 8, 14
7,990,842 ("'842 Patent")	1, 3, 4, 8, 11, 14, 19
6,941,156 ("'156 Patent")	1, 2
8,416,862 ("'862 Patent")	9, 10, 12
7,957,450 ("'450 Patent")	2, 3, 11, 12, 13, 21, 22

8,792,432 ("'432 Patent")	9, 12
7,039,435 ("'435 Patent")	1, 2, 3, 6, 8

b. Accused Instrumentalities (L.P.R. 3.1(b)):

Based on information presently available, BNR identifies the Accused Instrumentalities below. Defendants infringe the Asserted Claims pursuant to § 271(a) by making, using, importing, selling, and/or offering to sell in the United States without authority these Accused Instrumentalities. The particular acts constituting infringement by Defendants' accused instrumentalities are detailed in the claim charts attached herewith as Exhibits A - H. BNR reserves the right to amend or add additional Accused Instrumentalities to these Contentions, if warranted and/or based on further investigation and discovery.

Asserted Claim	Accused Instrumentalities
'889 Patent	
1	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
2	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
4	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
5	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
8	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.

9	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
12	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
’554 Patent	
1	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
2	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
4	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
5	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
7	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
8	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
14	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
’842 Patent	
1	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3

	8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
3	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
4	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
8	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
11	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
14	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
19	Ascend XT^2, Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
’156 Patent	
1	Mate 10 Pro, Ascend XT^2, Elate, Mate 9, Mate SE, Porsche Mate 10.

2	Mate 10 Pro, Ascend XT[2], Elate, Mate 9, Mate SE, Porsche Mate 10.
’862 Patent	
9	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
10	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
12	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
’450 Patent	
2	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
3	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
11	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
12	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
13	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
21	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
22	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
’432 Patent	
9	Mate 10 Pro, Mate 9, Porsche Mate 10, Honor View 10.
12	Mate 10 Pro, Mate 9, Porsche Mate 10, Honor View 10.
’435 Patent	

1	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
2	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
3	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
6	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
8	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.

c. Claim Charts for the Accused Instrumentalities (Patent L.R. 3.1(c)):

U.S. Patent No.	Claim Chart(s)
'889 Patent	*See* Exhibit A
'554 Patent	*See* Exhibit B
'842 Patent	*See* Exhibit C
'156 Patent	*See* Exhibit D
'862 Patent	*See* Exhibit E
'450 Patent	*See* Exhibit F
'432 Patent	*See* Exhibit G
'435 Patent	*See* Exhibit H

d. Indirect Infringement (Patent L.R. 3.1(d)):

The Asserted Claims are directly infringed by Defendants' Accused Instrumentalities under 35 U.S.C. § 271(a).

In addition to BNR's direct infringement contentions, BNR further contends that Defendants currently induce and/or have induced their customers to directly infringe the Asserted Claims by using the Accused Instrumentalities in the manner described in Exhibits A-H as instructed by the Defendants or by using the Accused Instrumentalities in the manner programmed by the Defendants.

BNR reserves the right to amend its pleadings and these Contentions to assert additional indirect infringement claims under § 271(b) or (c) and additional theories, if

warranted, based on BNR's ongoing investigation and discovery produced by Defendants.

e. Doctrine of Equivalents (Patent L.R. 3.1(e)):

At this time, BNR contends that each element of each Asserted Claim is literally present in each Accused Instrumentality as set forth above and identified in the exhibits and annexes hereto. BNR contends that if any element of the Asserted Claims is found to be not literally present in an Accused Instrumentality, that element is present under the Doctrine of Equivalents. The claim charts attached as exhibits hereto identify illustrative support for where the equivalent feature is found under the doctrine of equivalents pursuant to the function-way-result and/or insubstantial differences tests. BNR reserves the right amend or add to these contentions to identify additional contentions under the Doctrine of Equivalents based on discovery, BNR's investigation, and claim construction.

f. Priority Dates (Patent L.R. 3.1(f)):

BNR contends that each Asserted Claim is entitled to a priority date as indicated below in the chart, based on the date of application of the patent or a parent application. BNR makes this disclosure without waiving any argument that it is entitled to an earlier date of invention based on activities predating the filing of the applications indicated below.

U.S. Patent No.	Priority Date
'889 Patent claims	At least as early as June 17, 2003
'554 Patent claims	At least as early as June 17, 2003
'842 Patent claims	At least as early as July 27, 2004
'156 Patent claims	At least as early as June 26, 2001
'862 Patent claims	At least as early as April 21, 2005
'450 Patent claims	At least as early as December 14, 2004
'432 Patent claims	At least as early as February 14, 2011

’435 Patent claims	At least as early as September 28, 2001

g. BNR Products (Patent L.R. 3.1(g)):

At this time, BNR does not own an apparatus, product, device, process, method, act, or other instrumentality that practices the claimed inventions.

h. Willful Infringement (Patent L.R. 3.1(h)):

Defendants have infringed and are continuing to infringe the Asserted Claims in a manner that is willful, wanton, malicious, bad-faith, deliberate, consciously wrongful, and flagrant. Defendants’ egregious infringement is exemplified by the following facts:

- Defendants have known of their infringement of the ’889, ’554, ’842, ’156, ’862, and ’432 Patents since no later than December 1, 2017, when Mr. Afzal Dean, President of BNR, sent Defendants a letter identifying these Patents and notifying Defendants that their products (including examples thereof) infringe the Asserted Claims of each Patent. Mr. Dean offered to meet and present a detailed presentation to Defendants, describing the infringement.
- On January 18, 2018 and February 6, 2018, BNR followed up by sending additional letters regarding Defendants’ infringement of the Asserted Claims with respect to the ’889, ’554, ’842, ’156, ’862, and ’432 Patents.
- BNR participated in meetings with Huawei on or about March 16, 2018, April 23, 2018, and June 20, 2018 in Shenzhen, China, to discuss Defendants’ infringement of the ’889, ’554, ’842, ’862, and ’432 Patents.
- On September 14, 2018, Huawei executives met with BNR executives in person and discussed, *inter alia*, Huawei’s infringement of the ’889, ’554, ’842, ’862, and ’432 Patents.
- Defendants have known of their infringement of the ’450 and ’435 Patents since no later than November 13, 2018, when the Second Amended Complaint in this action was filed.

BNR expressly reserves the right to amend or add to its willfulness contentions as more information comes to light through its investigation and discovery in this matter.

Dated: December 7, 2018

/s/ *Sadaf R. Abdullah*

Mieke K. Malmberg
(SBN 209992)
SKIERMONT DERBY LLP
800 Wilshire Blvd., Ste. 1450
Los Angeles, CA 90017
Phone: (213) 788-4500
Fax: (213)788-4545
mmalmberg@skiermontderby.com

Paul J. Skiermont (*pro hac vice*)
Sadaf R. Abdullah (*pro hac vice*)
Steven W. Hartsell (*pro hac vice*)
Christopher M. Hodge (*pro hac vice*)
Steven J. Udick* (TX Bar No. 24079884)
SKIERMONT DERBY LLP
1601 Elm St., Ste. 4400
Dallas, TX 75201
Phone: (214) 978-6600
Fax: (214) 978-6601
pskiermont@skiermontderby.com
sabdullah@skiermontderby.com
shartsell@skiermontderby.com
chodge@skiermontderby.com
sudick@skiermontderby.com
(* denotes pro hac vice to be filed)

Attorneys for Plaintiff
BELL NORTHERN RESEARCH, LLC

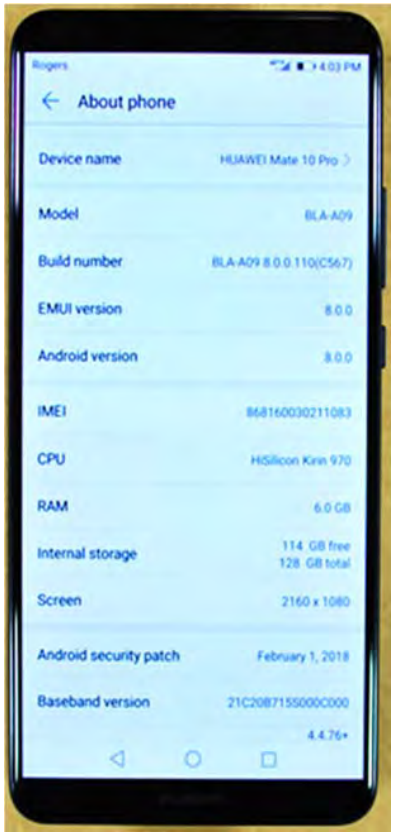
CERTIFICATE OF SERVICE

I hereby certify that on December 7, 2018, I served the foregoing document via email to the following counsel of record for the Defendants:

Jason W. Wolff
Joanna M. Fuller
Robert M. Yeh
wolff@fr.com
jfuller@fr.com
ryeh@fr.com

/s/ *Sadaf R. Abdullah*

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Mate 10 Pro and Porsche Design Mate 10[1]
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Mate 10 Pro is a mobile station.  

[1] The Huawei Mate 10 Pro and the Huawei Porsche Design Mate 10 share all relevant hardware components and software functionalities. The only internal difference is that the Porsche Design Mate 10 includes more on-board storage; but this difference does not

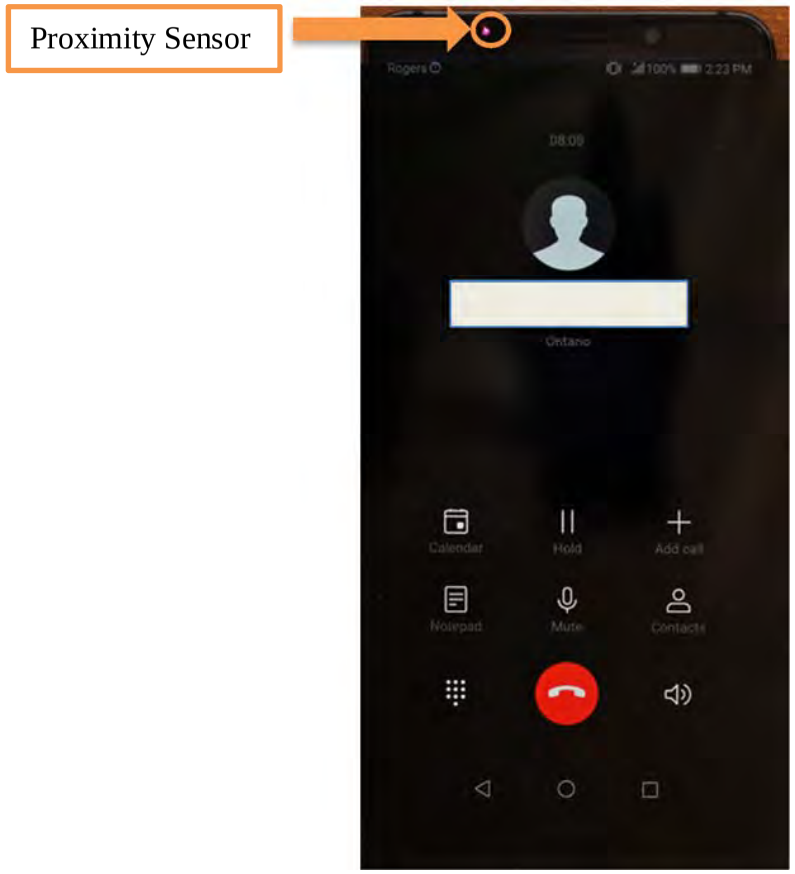
Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Mate 10 Pro includes a display.  *See https://consumer.huawei.com/uk/support/phones/mate10-pro/ p. 1, last accessed Nov 21, 2018.*

15

relate to the accused functionalities. *See* https://www.phonearena.com/news/Huawei-Mate-10-Pro-Mate-10-Porsche-Design-differences_id102298, *last accessed December 5, 2018.*

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Mate 10 Pro includes a proximity sensor. 

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Mate 10 Pro is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Mate 10 Pro is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

	The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more. **Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

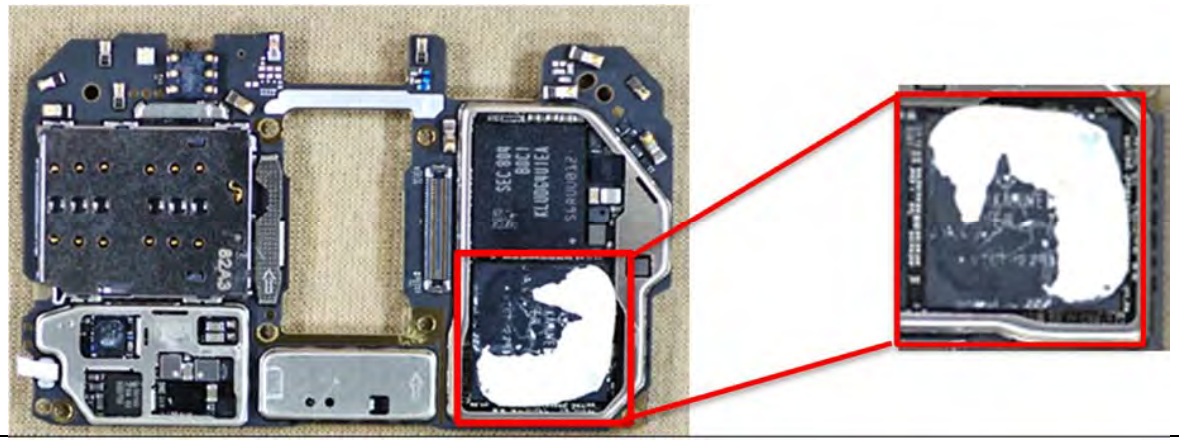
19	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

20

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Mate 10 Pro includes at least one microprocessor. 

	"**Chipset:** Huawei Kirin 970" *See https://consumer.huawei.com/en/phones/mate10-pro/specs/, last accessed Nov 21, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate 10 Pro is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.

22

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

23	  By way of example only, the Huawei Mate 10 Pro is an Android phone. The Huawei Mate 10 Pro's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call:

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

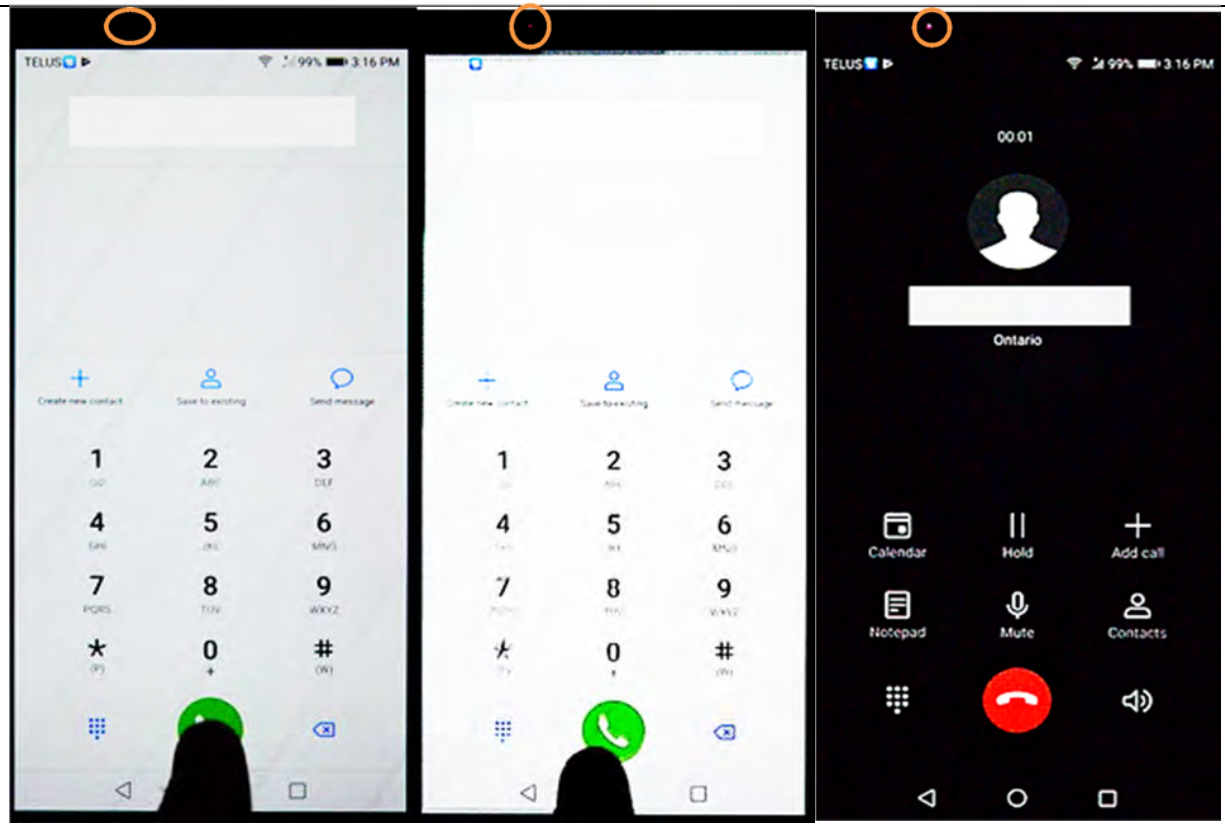
In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Mate 10 Pro is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

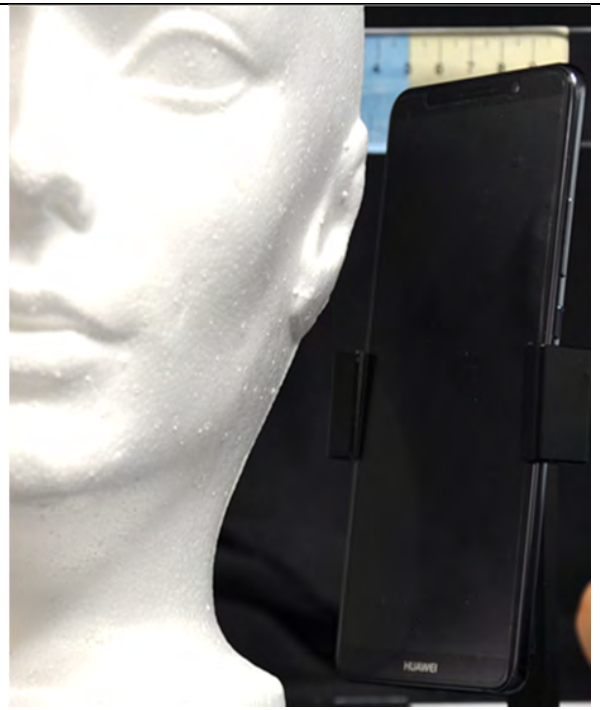
27	TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Mate 10 Pro, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Mate 10 Pro is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

28

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	 By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

29

30

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

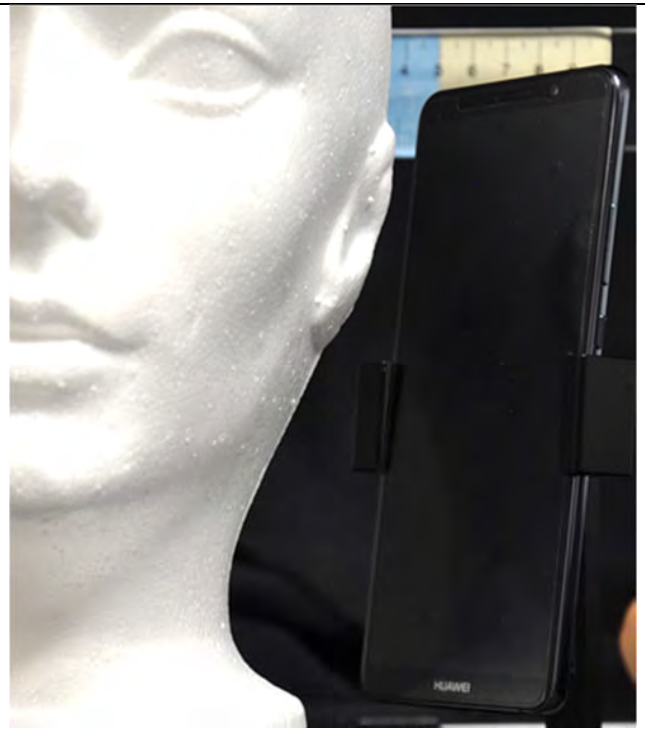
Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

31

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Mate 10 Pro is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Mate 10 Pro is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is darkened.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

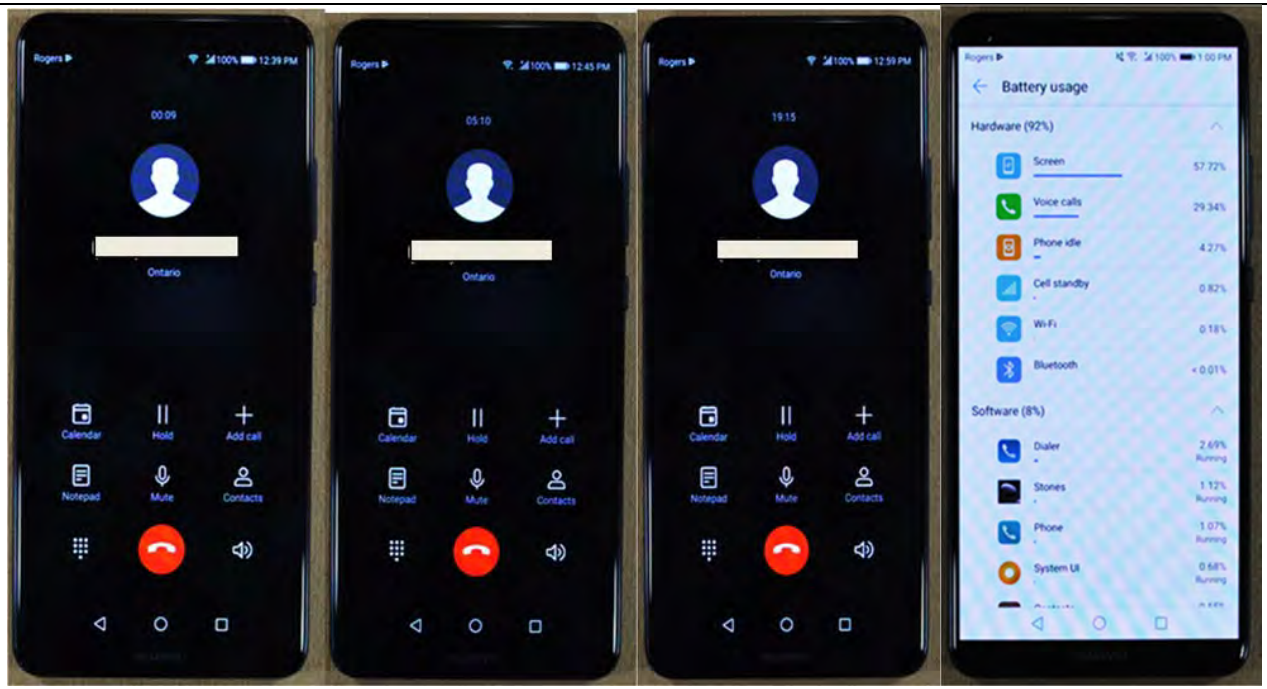
33

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

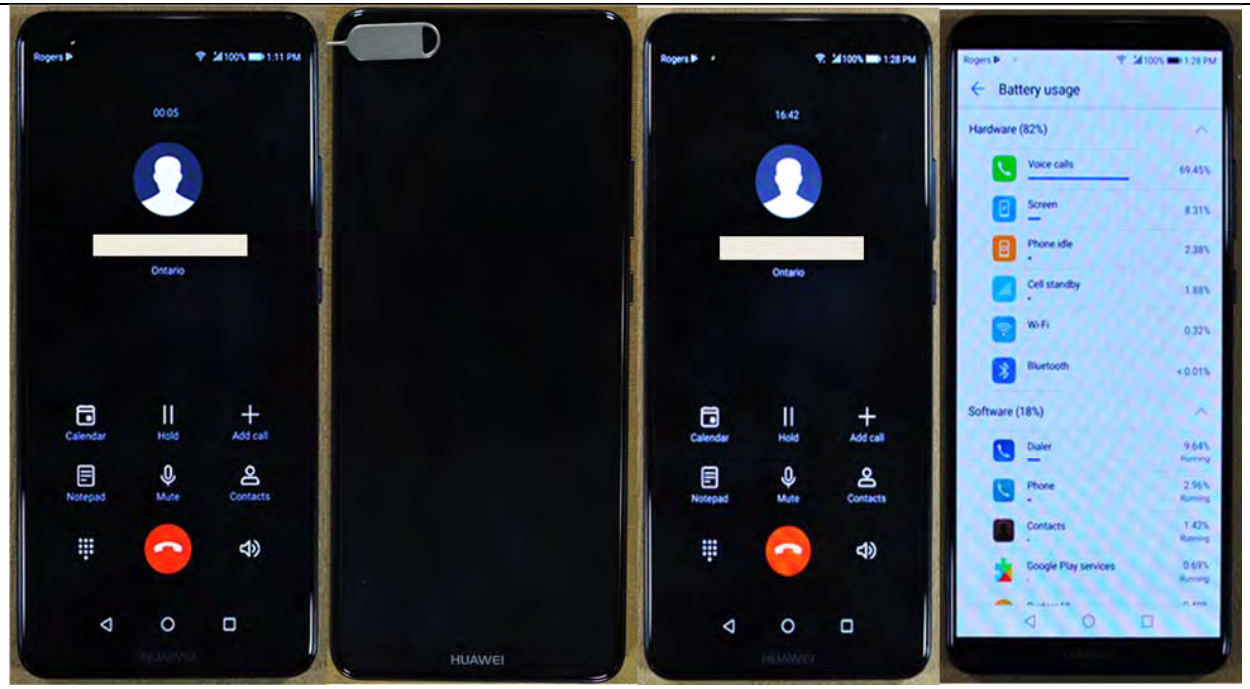
By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.72% of the battery usage. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

35

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 8.31% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

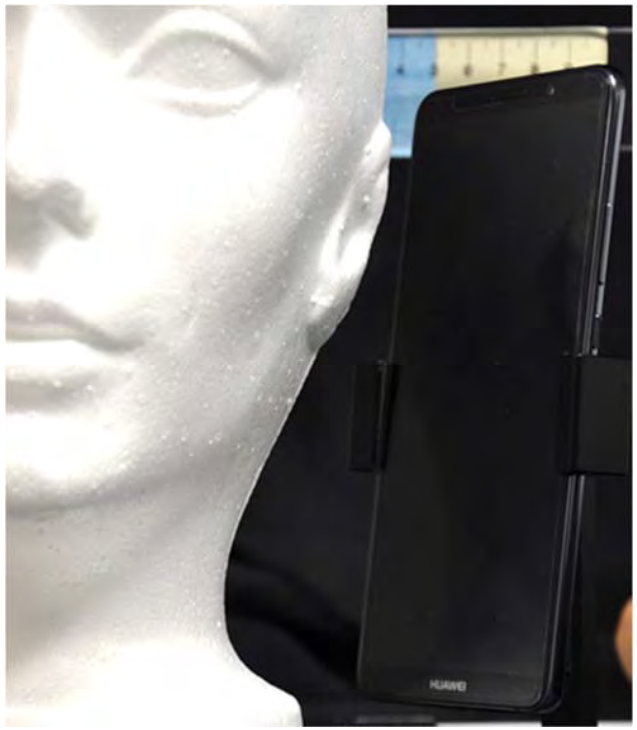
	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

38

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The microprocessor of the Huawei Mate 10 Pro increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Mate 10 Pro is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is darkened.

40

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

41

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

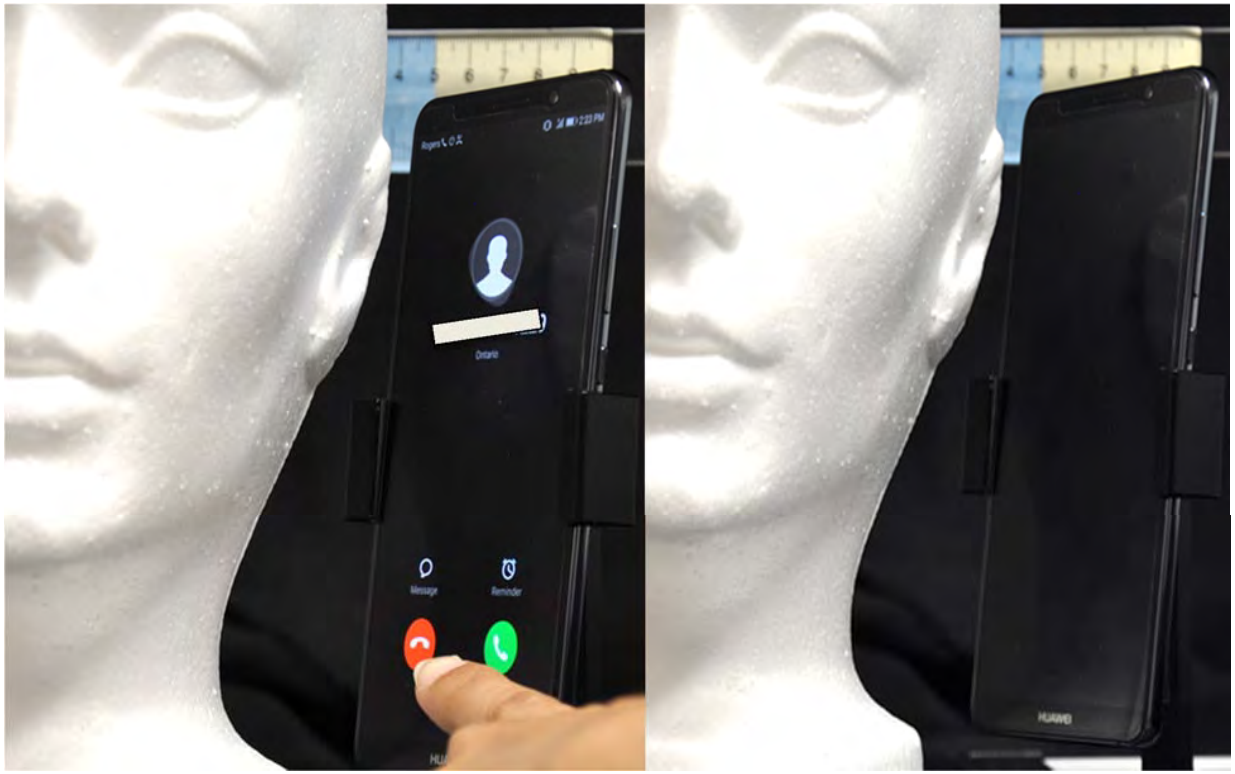
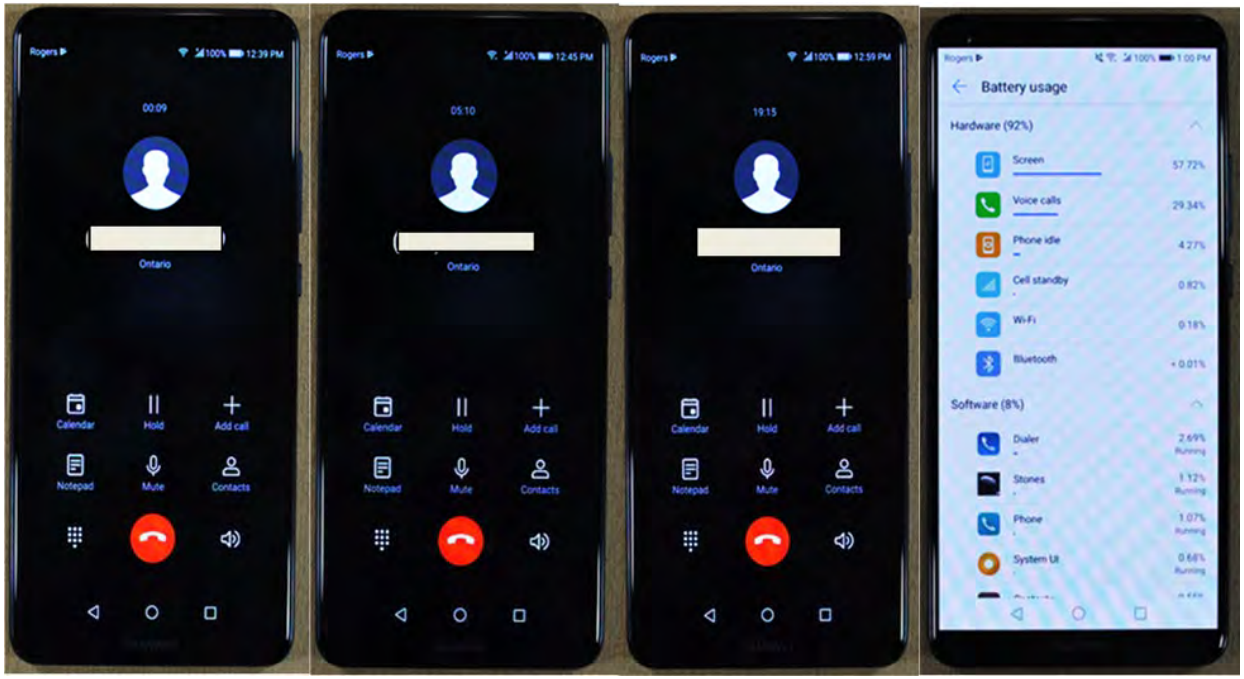
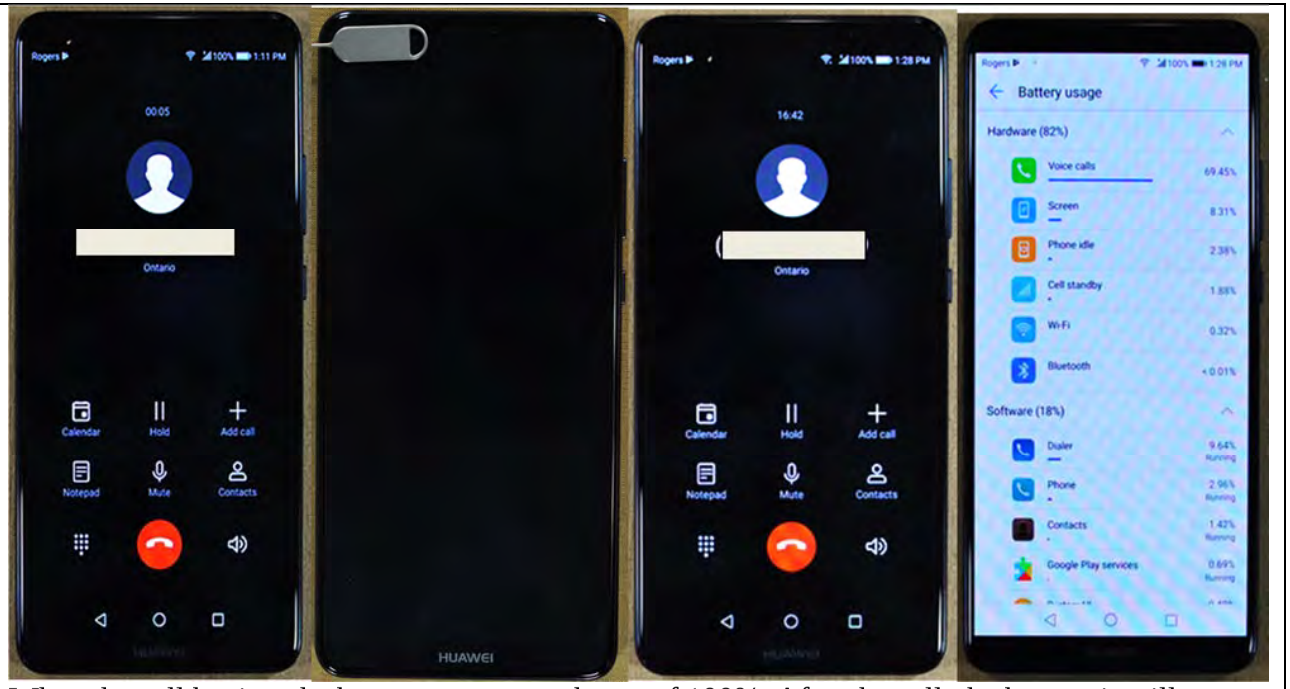
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Mate 10 Pro reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally  When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.72% of the battery usage. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

44

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 8.31% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

45

46

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

47

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

48

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

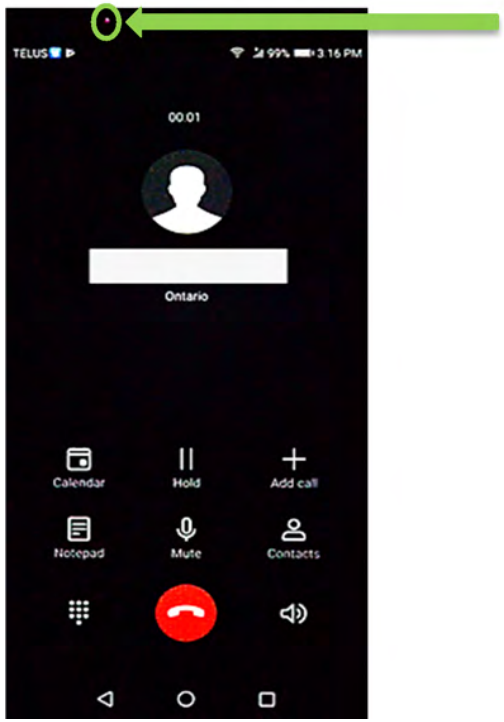
	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

49

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Mate 10 Pro is a range-detecting sensor.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

51	The proximity sensor detects the range of a proximate object:  *The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate* By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.

By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

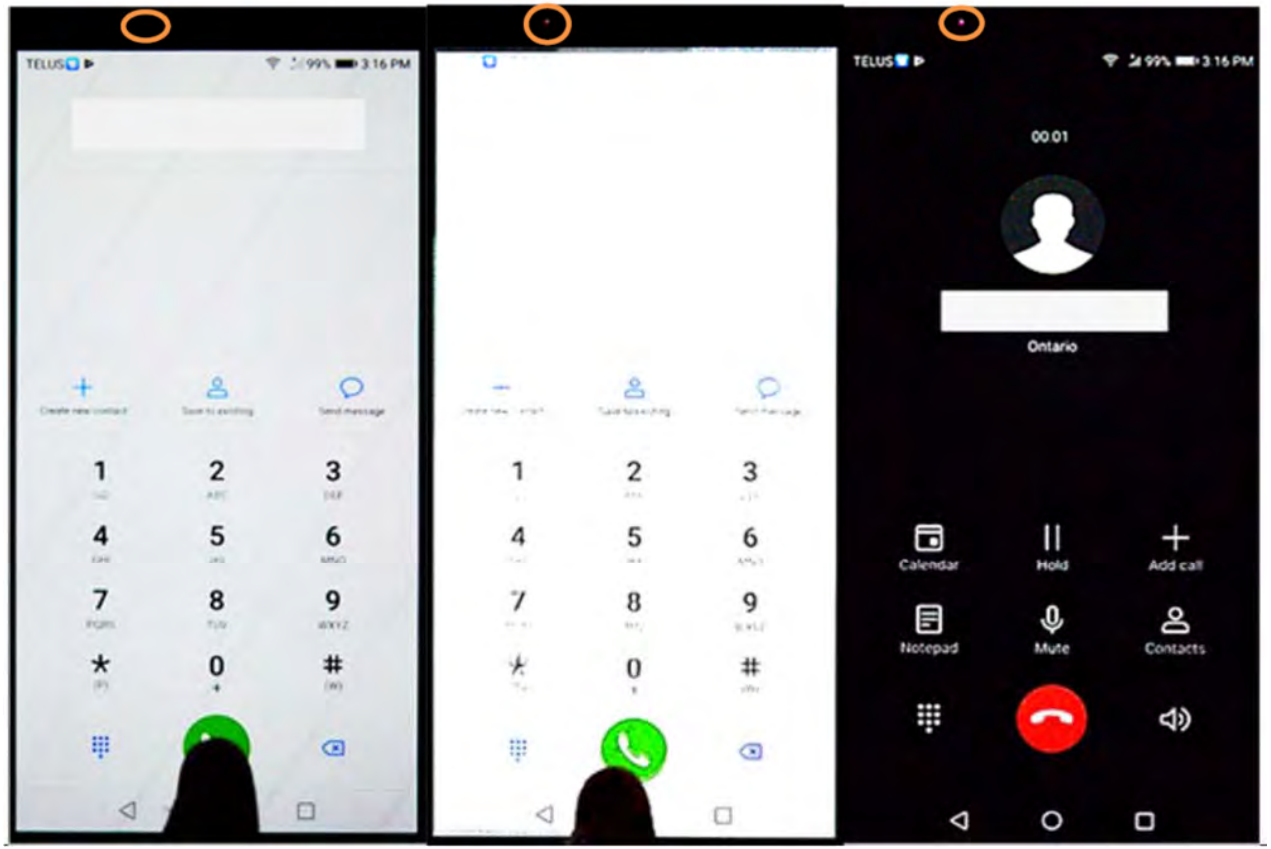
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.* The Android Developer Code Website further explains: **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method. *See id.*

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Mate 10 Pro begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

55



In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

56

Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

57

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Mate 10 Pro detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

58

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Mate 10 Pro activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Mate 10 Pro reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate 10 Pro is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

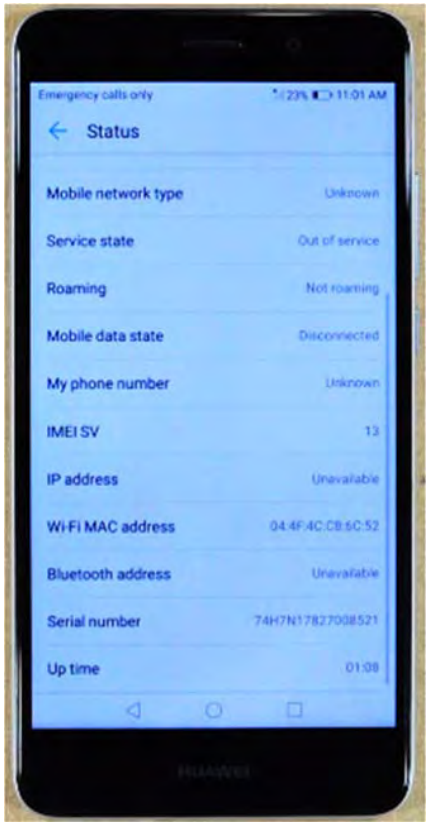
Asserted Claim Elements	Huawei Ascend XT2
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Ascend XT2 is a mobile station.  

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Ascend XT2 includes a display.  *See http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350616, last accessed Nov 30, 2018.*

[ii] a proximity sensor	The Huawei Ascend XT2 includes a proximity sensor.  *See http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350616, last accessed Nov 30, 2018.*

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Ascend XT2 is adapted to generate a signal indicative of existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Ascend XT2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

65	The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more. **Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

99	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

67	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

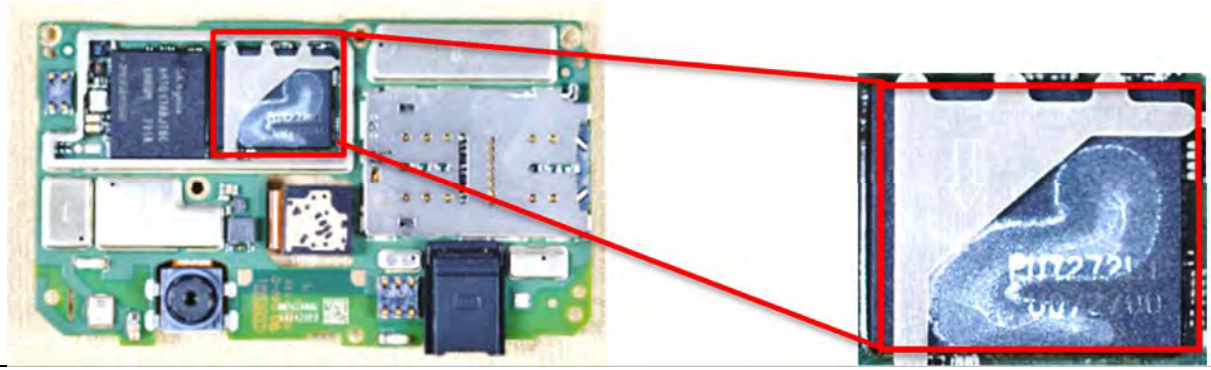
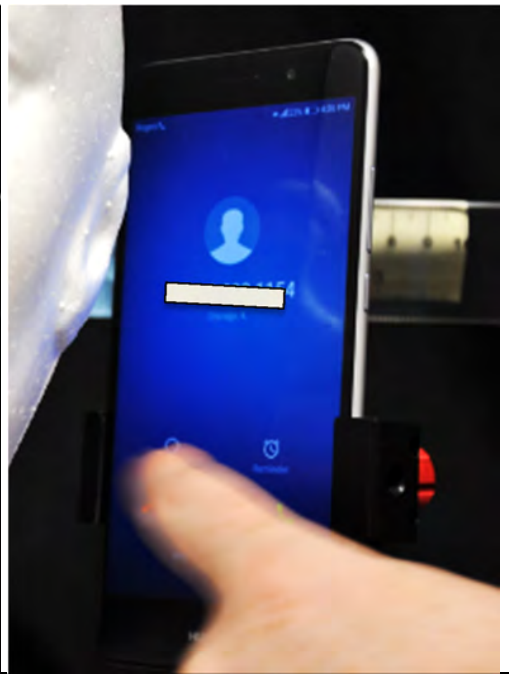
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Ascend XT2 includes at least one microprocessor. 

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

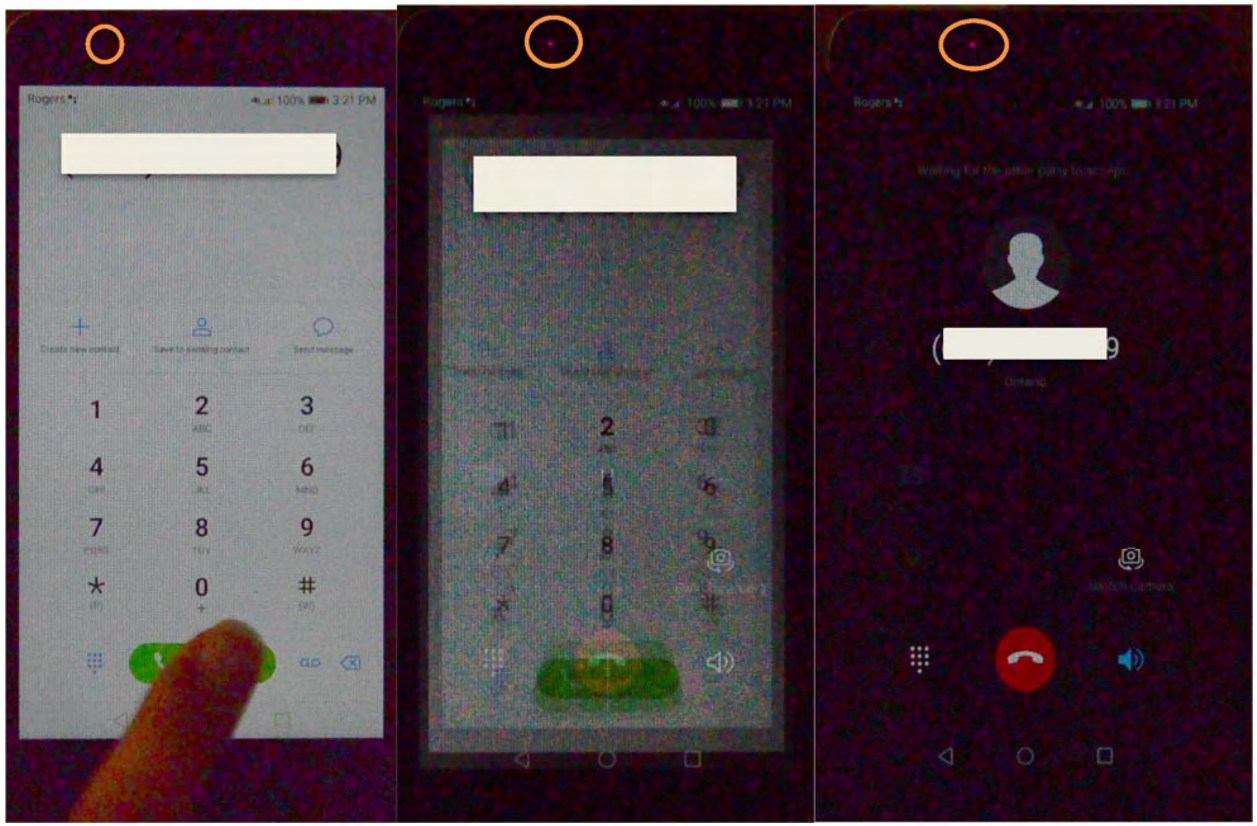
	"**Chipset:** Qualcomm Snapdragon 435 (MSM8940) Octa-Core (4 x 1.4 GHz+4 x 1.1 GHz)" *See https://consumer.huawei.com/us/phones/ascend-xt2/specs/, last accessed Nov 30, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Ascend XT2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.  

69

70	By way of example only, the Huawei Ascend XT2 is an Android phone. The Huawei Ascend XT2's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `public OutgoingReceiver() {` `}` `@Override` `public void onReceive(Context context, Intent intent) {` `String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `Toast.makeText(ctx,` `"Outgoing: "+number,` `Toast.LENGTH_LONG).show();` `}` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Ascend XT2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

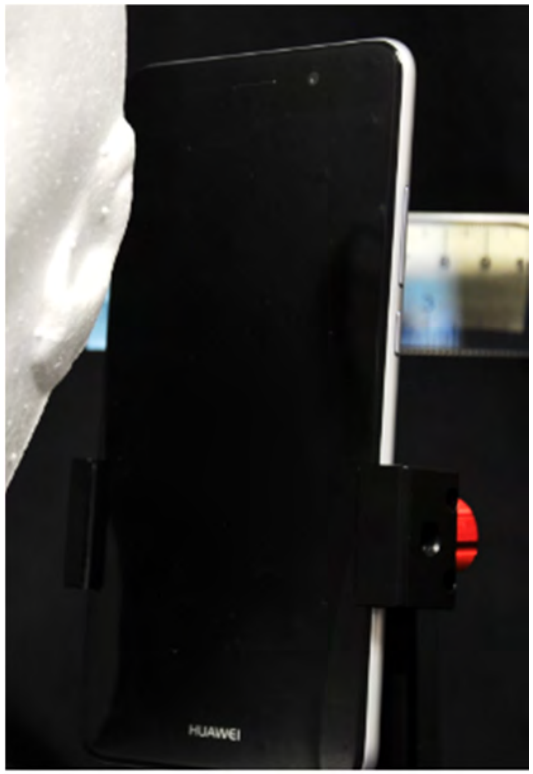
	

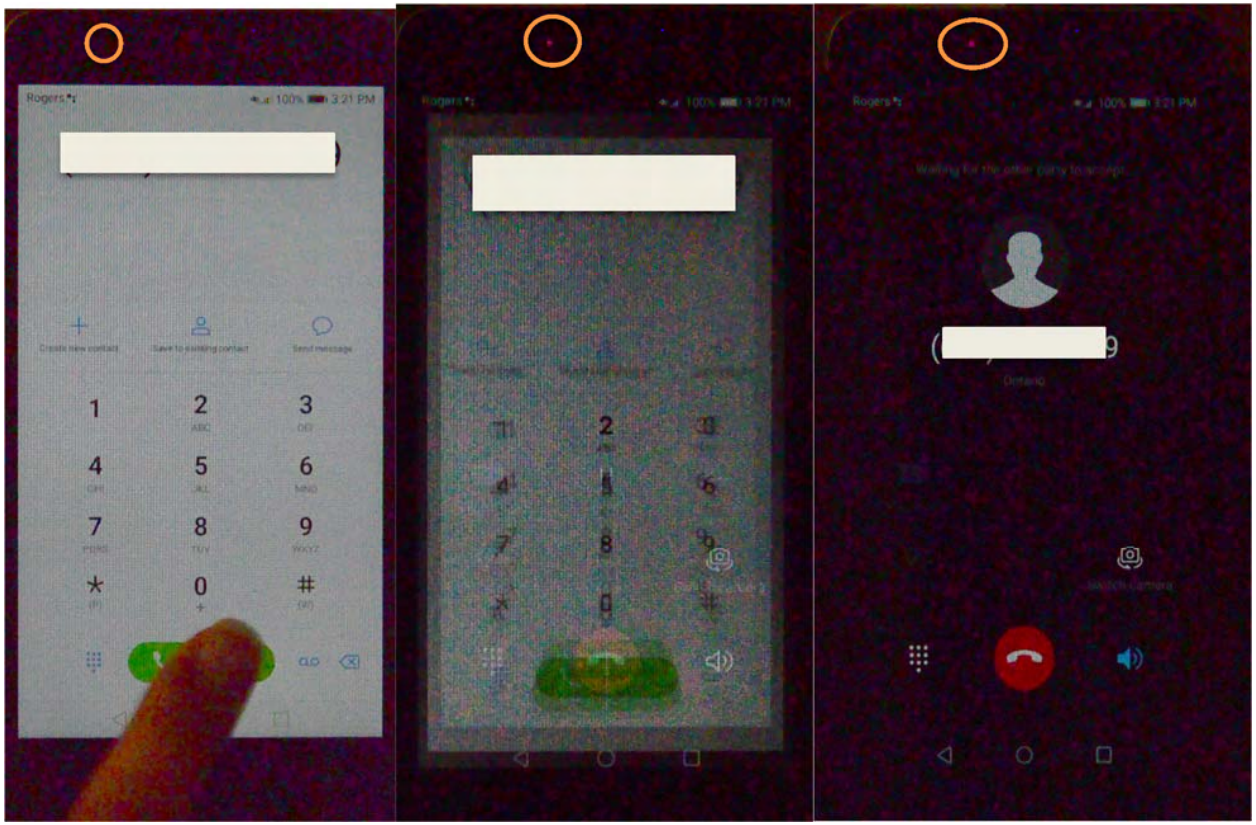
	By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use: TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more. **Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Ascend XT2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Ascend XT2 is adapted to receive the signal from the activated proximity sensor.

74

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Ascend XT2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.  The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is

	activated. Third image shows the proximity sensor remains activated during the remainder of the call.  By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

77

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

78

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The microprocessor of the Huawei Ascend XT2 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Ascend XT2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend XT2, power to the display is reduced and the display is darkened.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

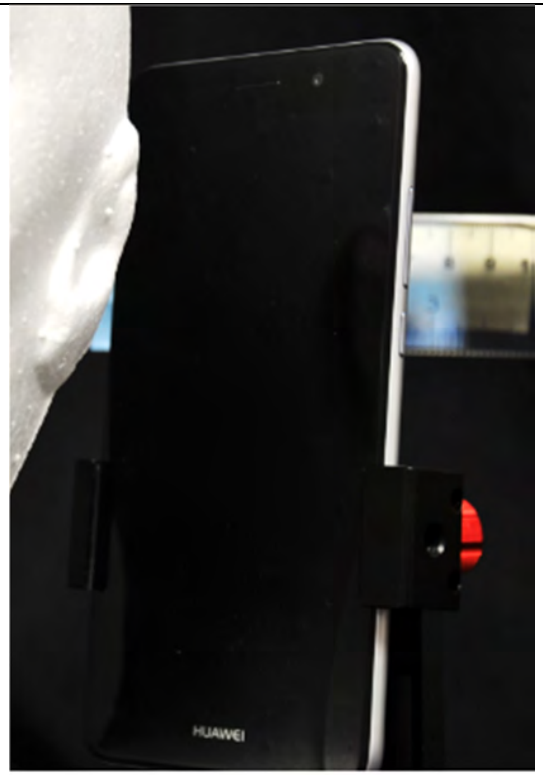
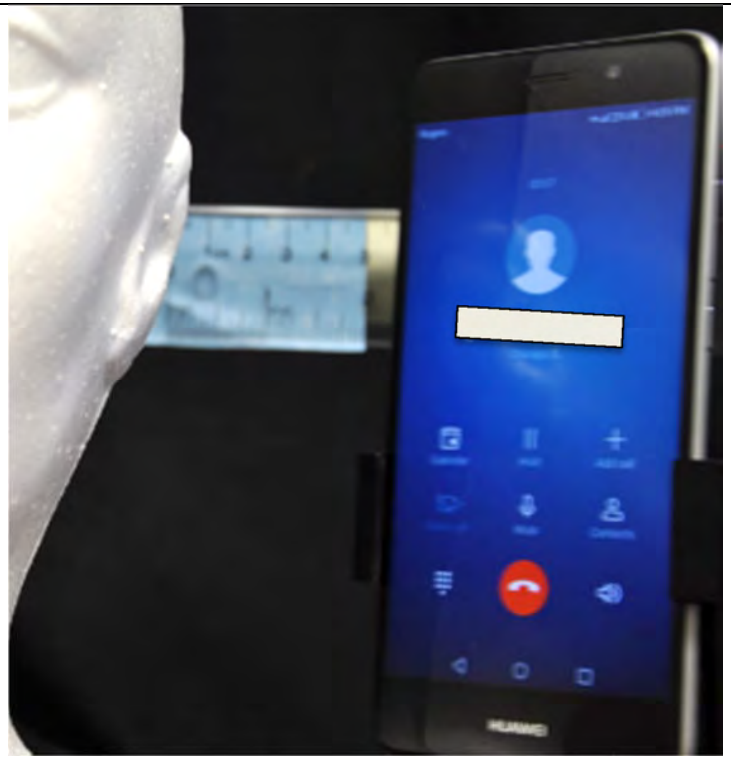
80	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

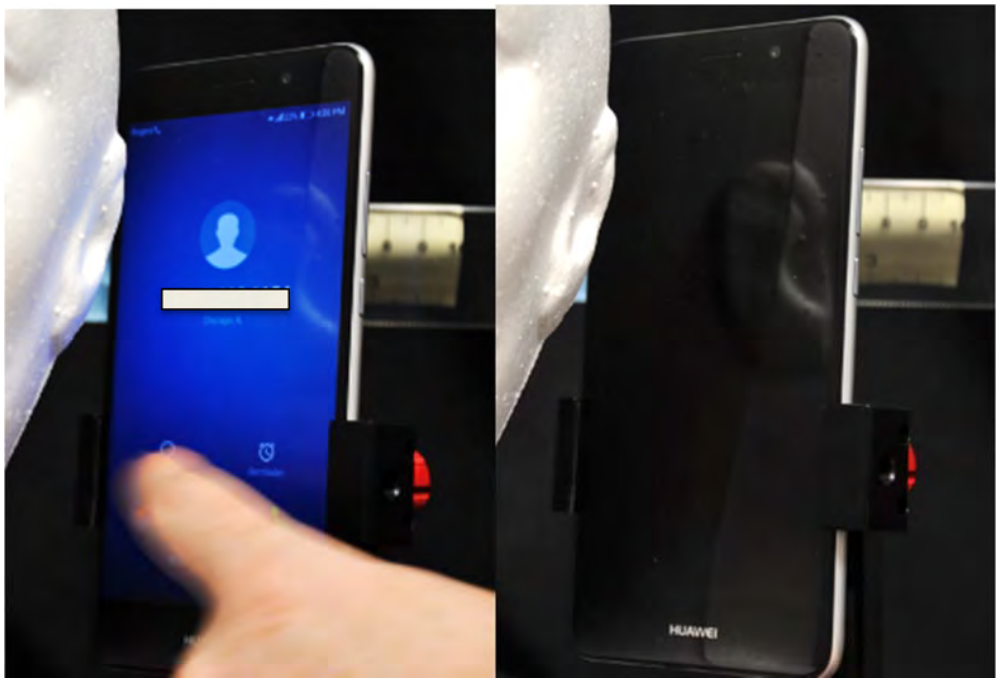
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Ascend XT2 reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Ascend XT2, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

83	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 97%. The screen consumed 49.32% of the battery. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

84	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.27% of the battery. By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

85

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

86

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

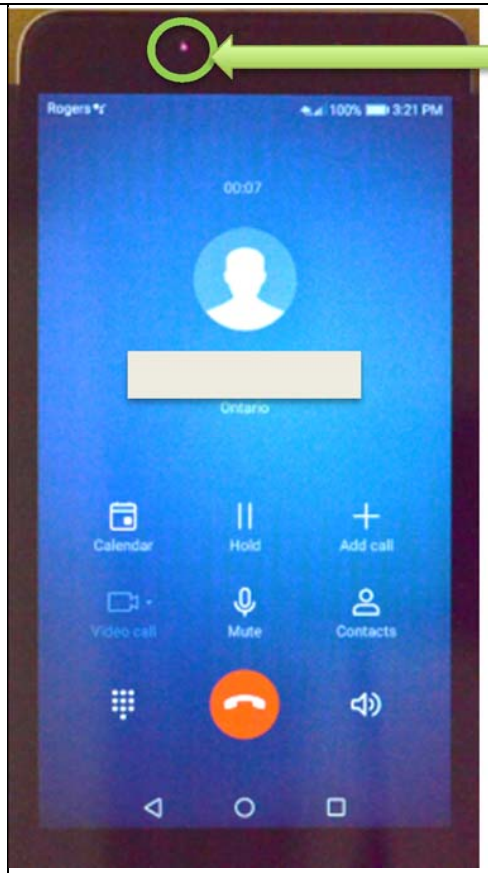
87	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

88	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Ascend XT2 is a range-detecting sensor.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

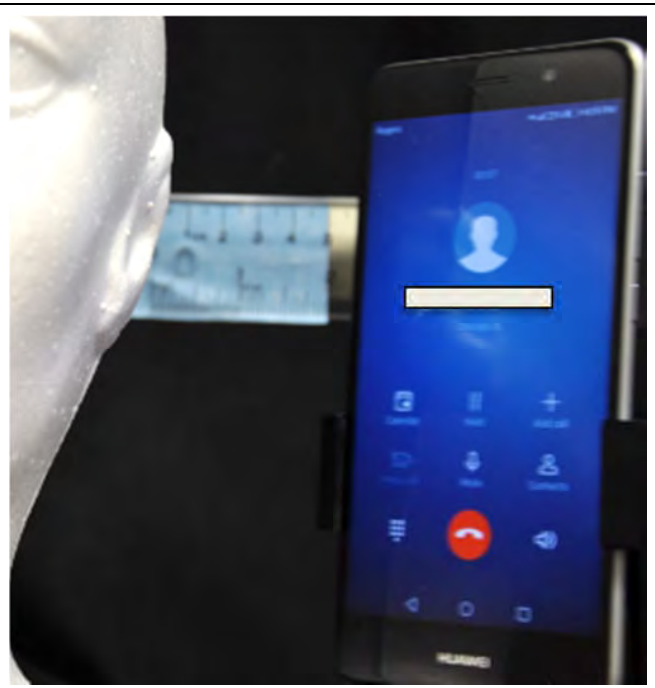


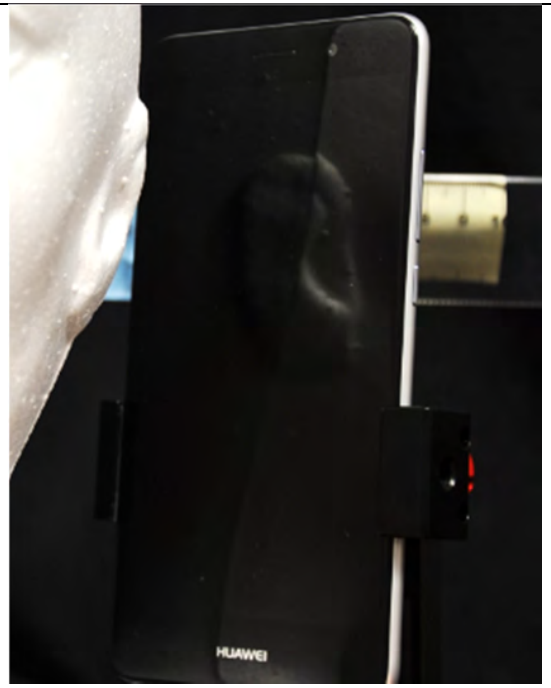
The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

The proximity sensor detects the range of a proximate object:

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.

90





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

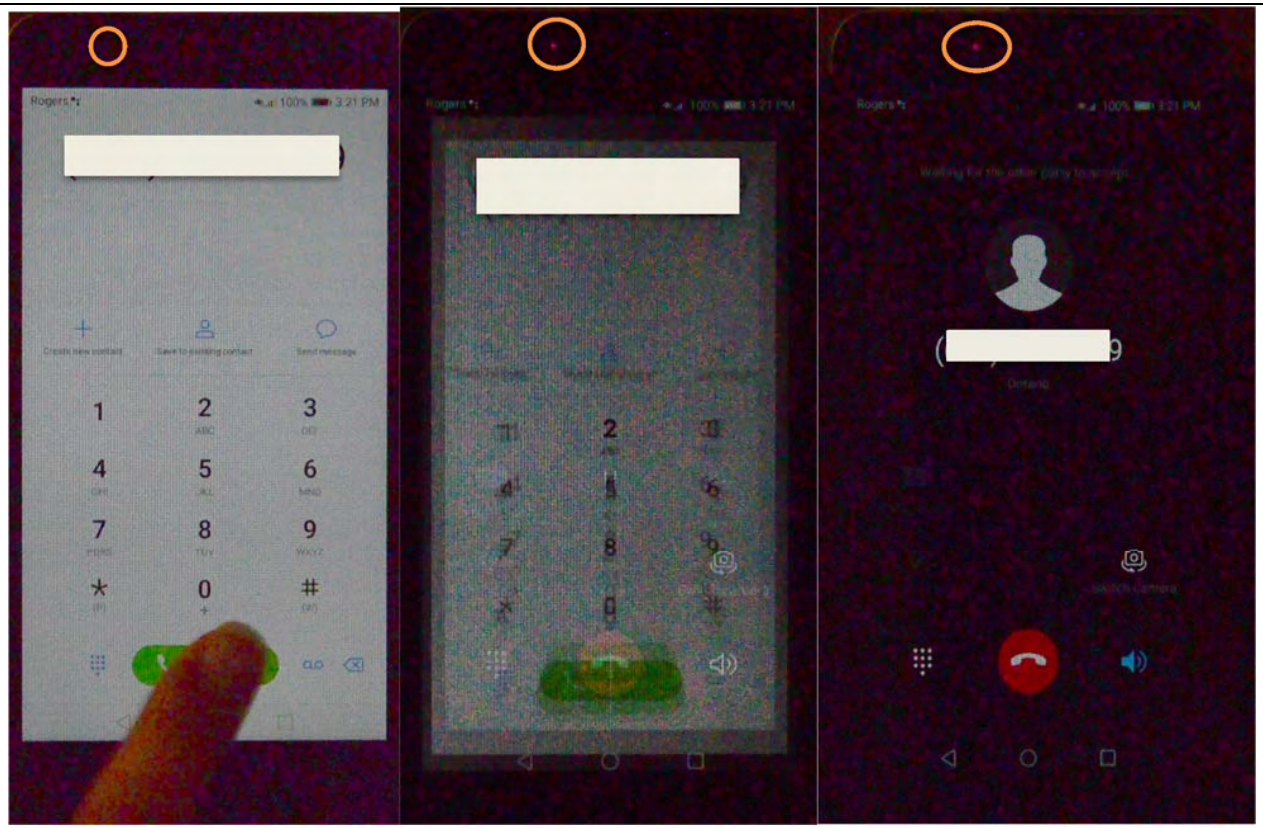
92

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Ascend XT2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

94



In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

95	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

96

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different	The Huawei Ascend XT2 T detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile

97

from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Ascend XT2 T activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Ascend XT2 T reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].
and a microprocessor adapted to:	*See* claim[iv].

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Ascend XT2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

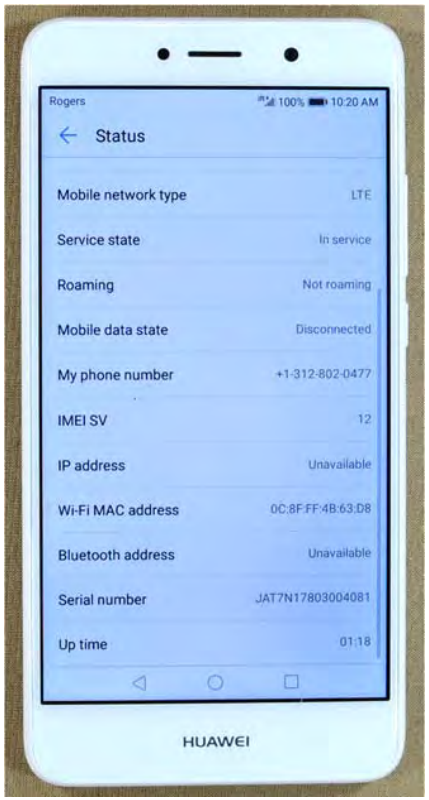
Asserted Claim Elements	Huawei Elate
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Elate is a mobile station.  

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Elate includes a display.  *See https://www.cricketwireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf, last accessed Nov 22, 2018.*

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Elate includes a proximity sensor.  *See https://www.cricketwireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf, last accessed Nov 22, 2018.*

102

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

103	 *See* https://www.cricketwireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf, p. 8 (dated 2017; last accessed Nov 22, 2018)

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Elate is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Elate is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

105

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

106	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

107	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

108

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

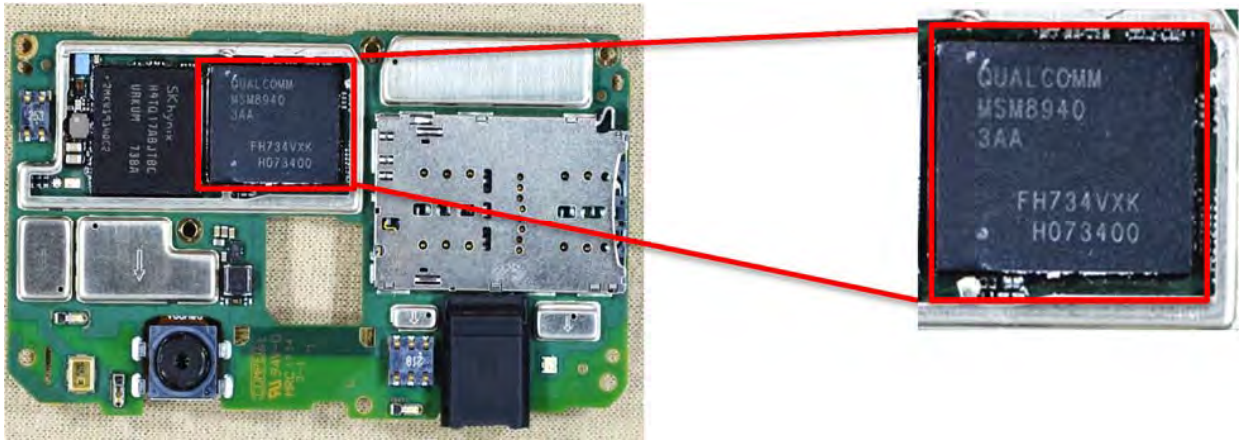
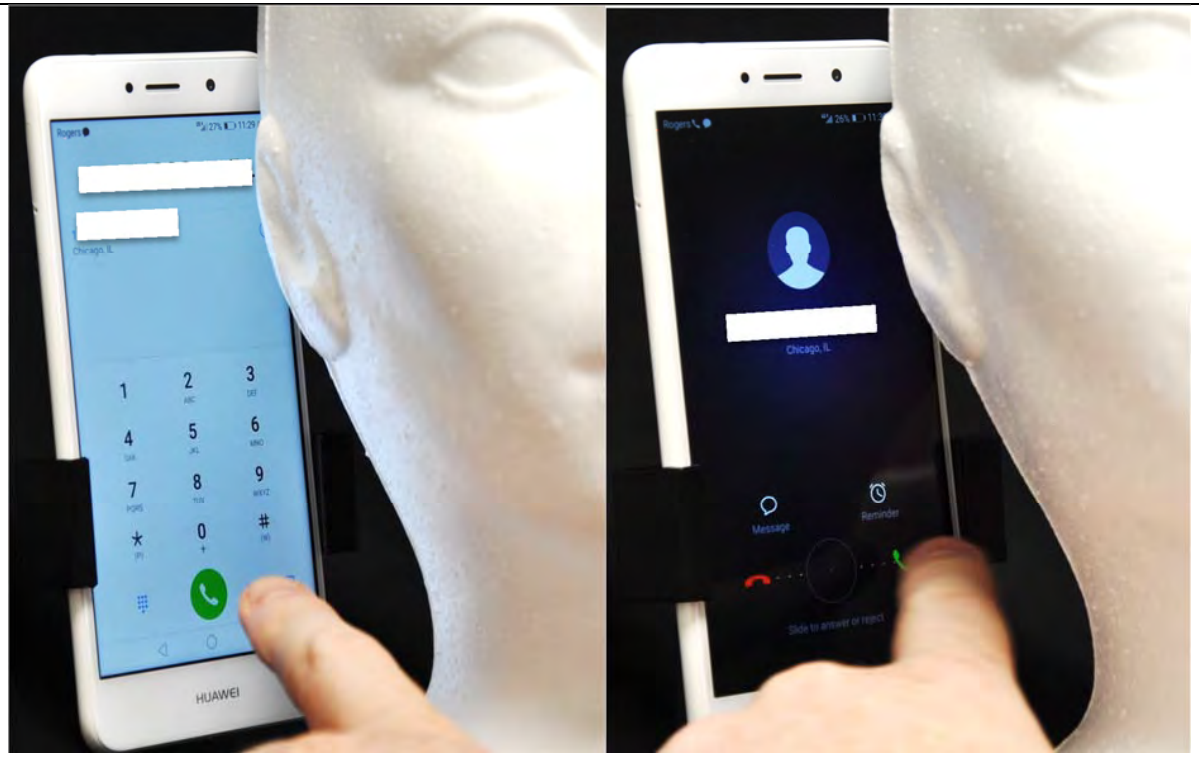
[iv] a microprocessor adapted to:	The Huawei Elate includes at least one microprocessor.  "**Chipset:** Qualcomm Snapdragon 435 (MSM8940) Octa-Core (4 x 1.4 GHz+4 x 1.1 GHz) " *See* *https://consumer.huawei.com/us/phones/elate/specs/* (last accessed Nov 22, 2018)
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Elate is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

110	 By way of example only, the Huawei Elate is an Android phone. The Huawei Elate's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call:

111

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

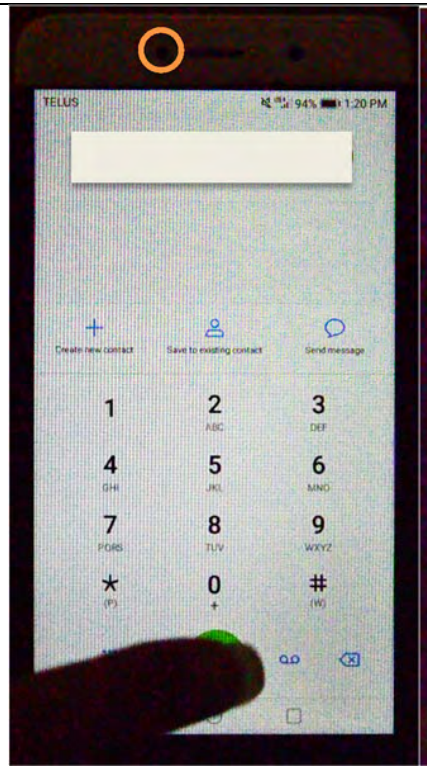
```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

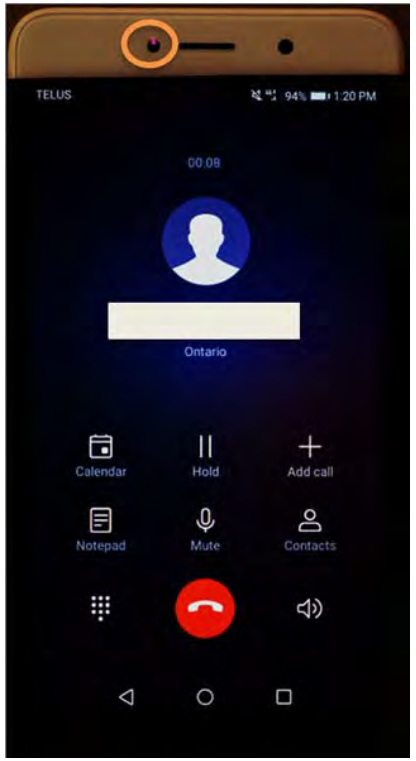
In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Elate is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

112

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554





By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

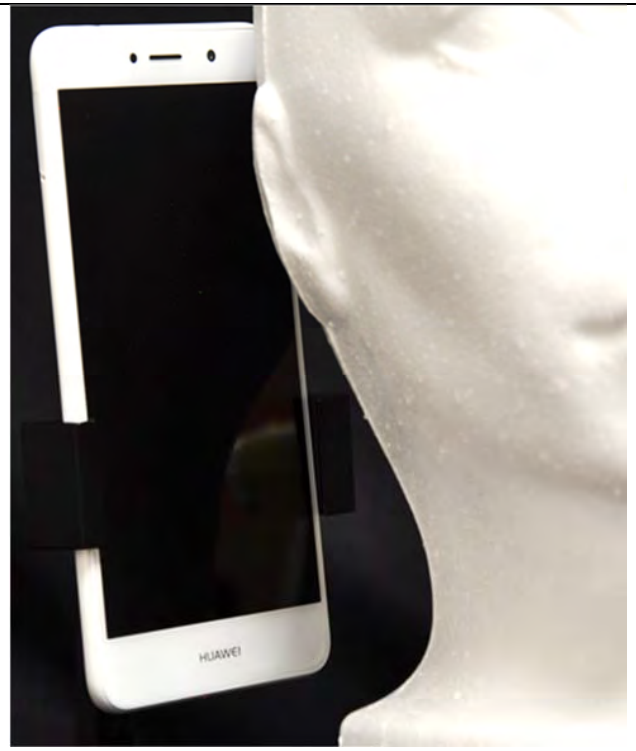
114	The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use: TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Elate, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Elate is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

115

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

	 By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

116

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

117

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

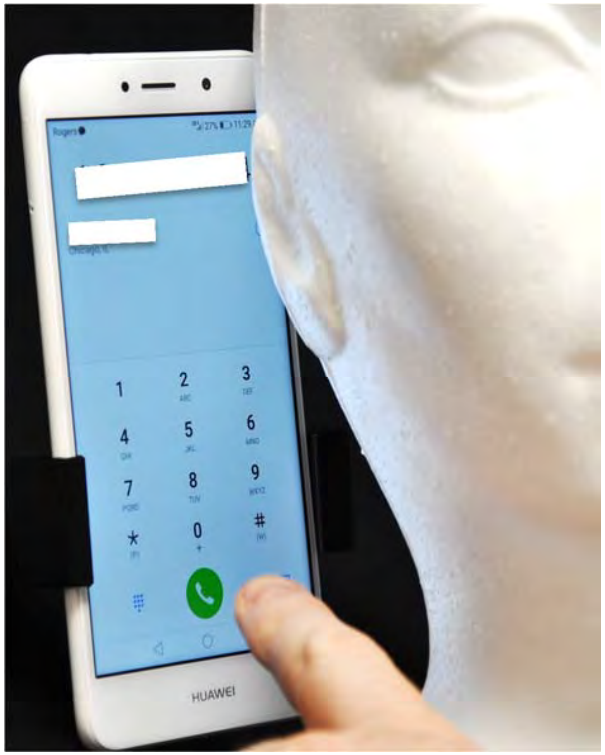
Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

118

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Elate is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Elate is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is darkened.

119

120	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

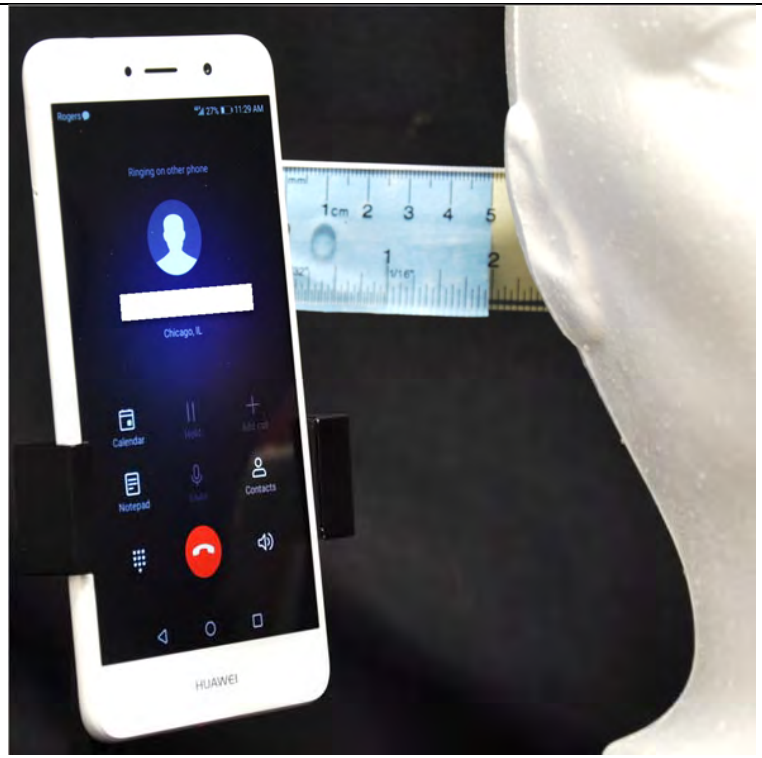
121	 The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

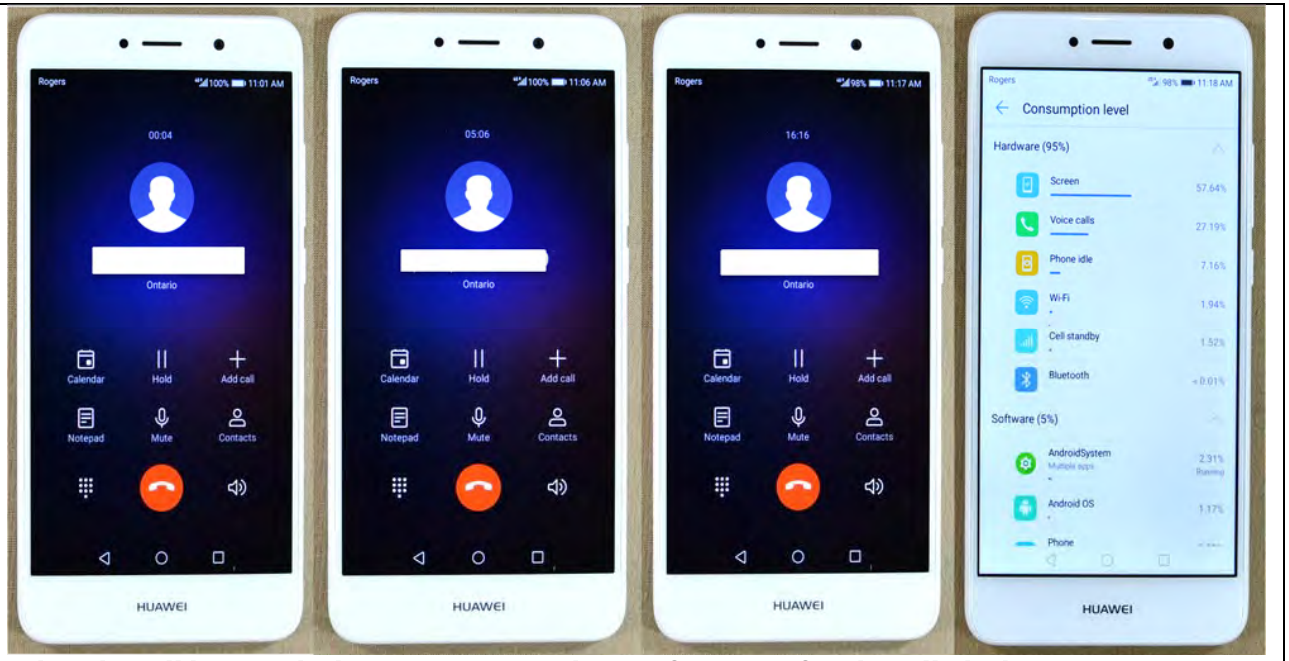
122	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 57.64% of the battery consumption. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.17% of the battery consumption.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

124

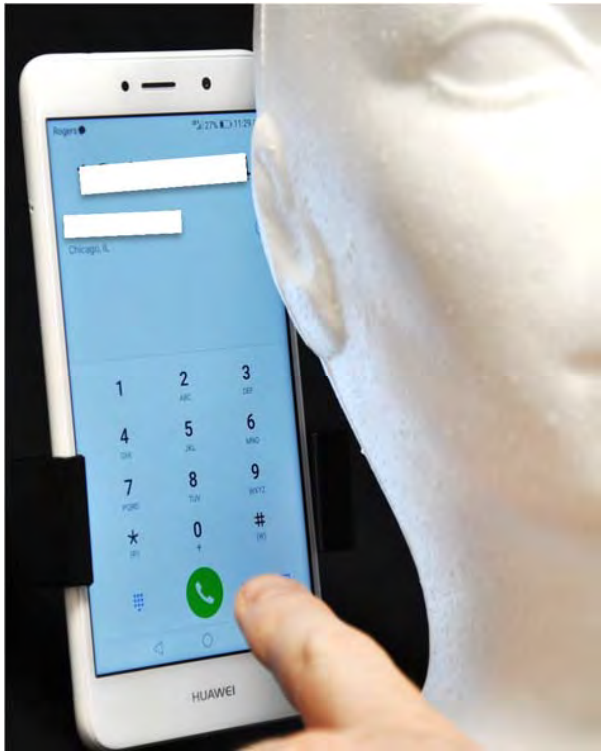
Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

125	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

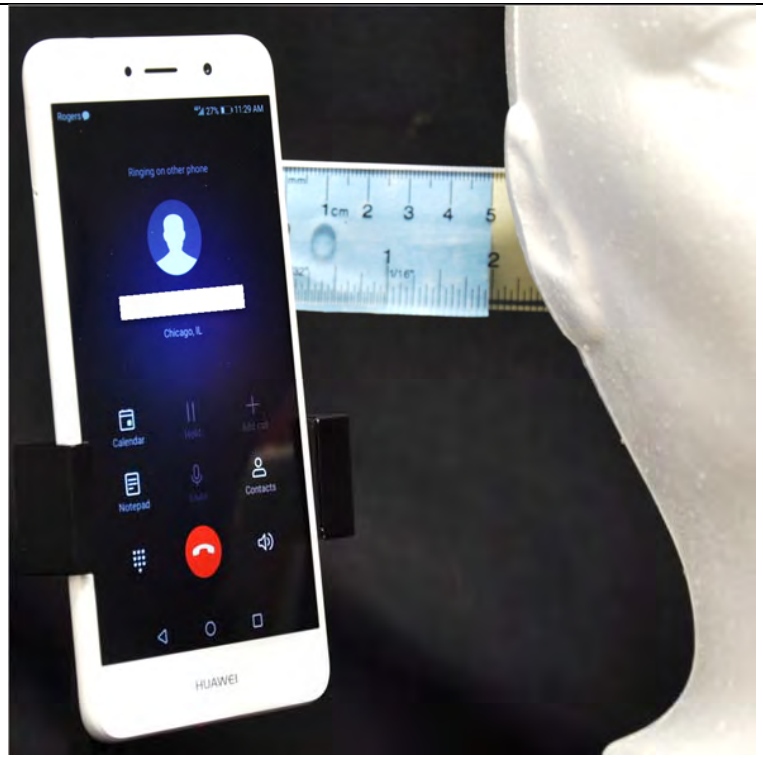
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018*
2. The mobile station of claim 1,	*See* claim 1.

126

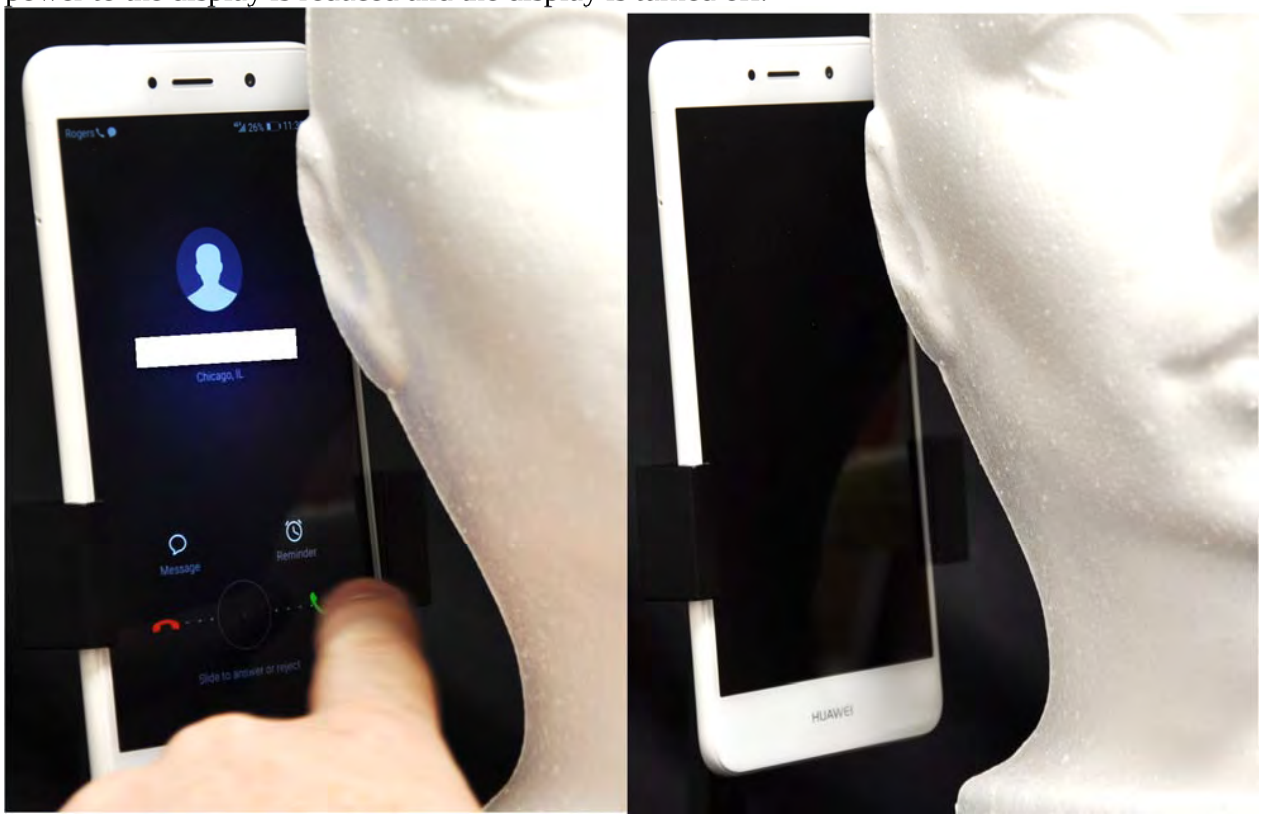
further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The microprocessor of the Huawei Elate increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Elate is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is darkened.

128	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

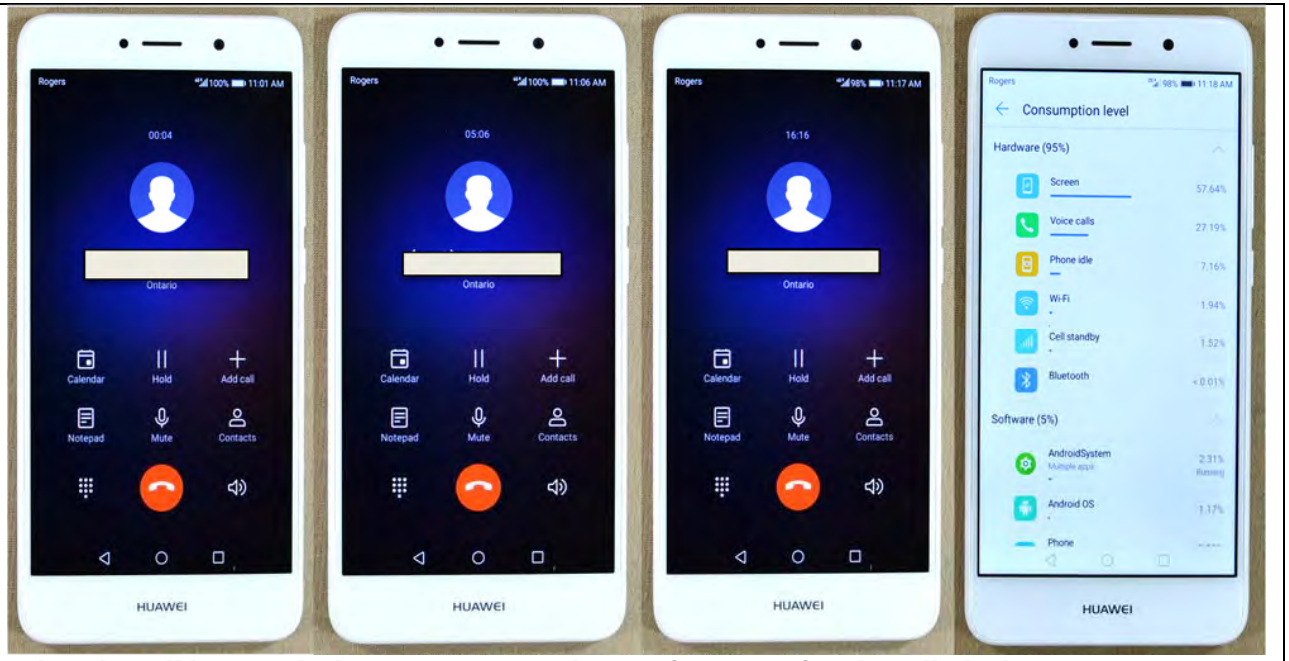
129

wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Elate reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

130

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

131



When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 57.64% of the battery consumption.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

132

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.17% of the battery consumption.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

133

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

134

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

135	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

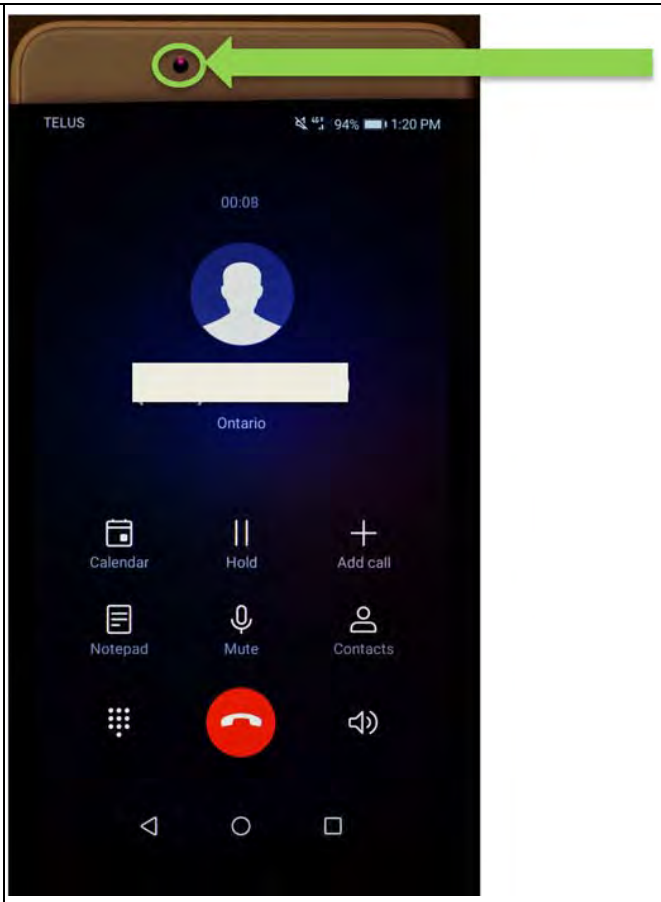
136	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Elate is a range-detecting sensor.

137

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

138

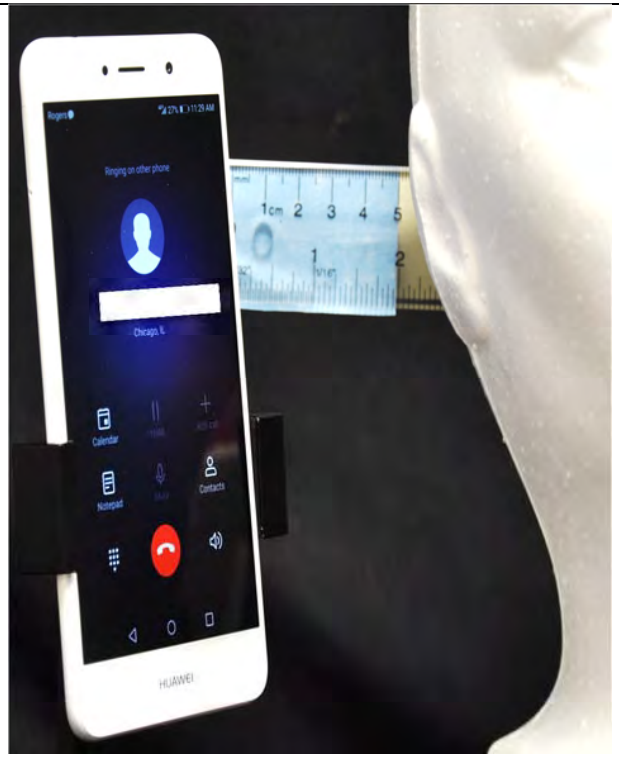


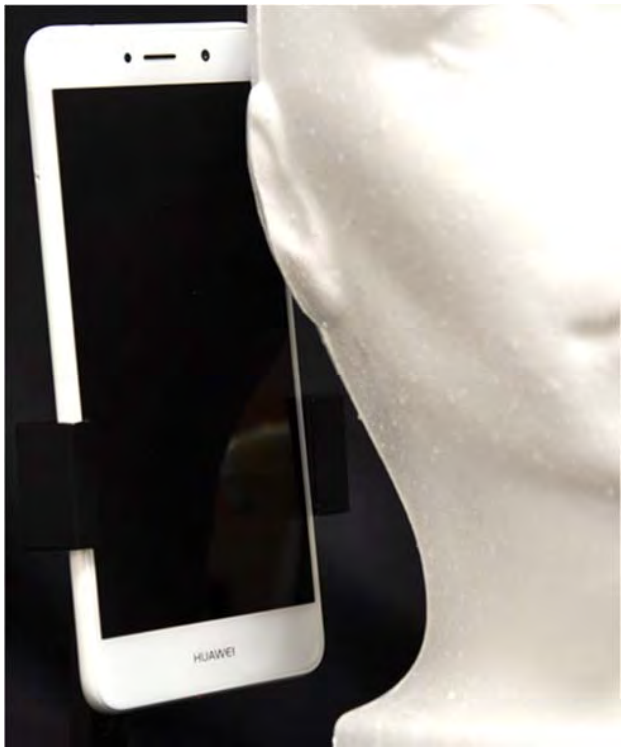
The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

The proximity sensor detects the range of a proximate object:

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.

139





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

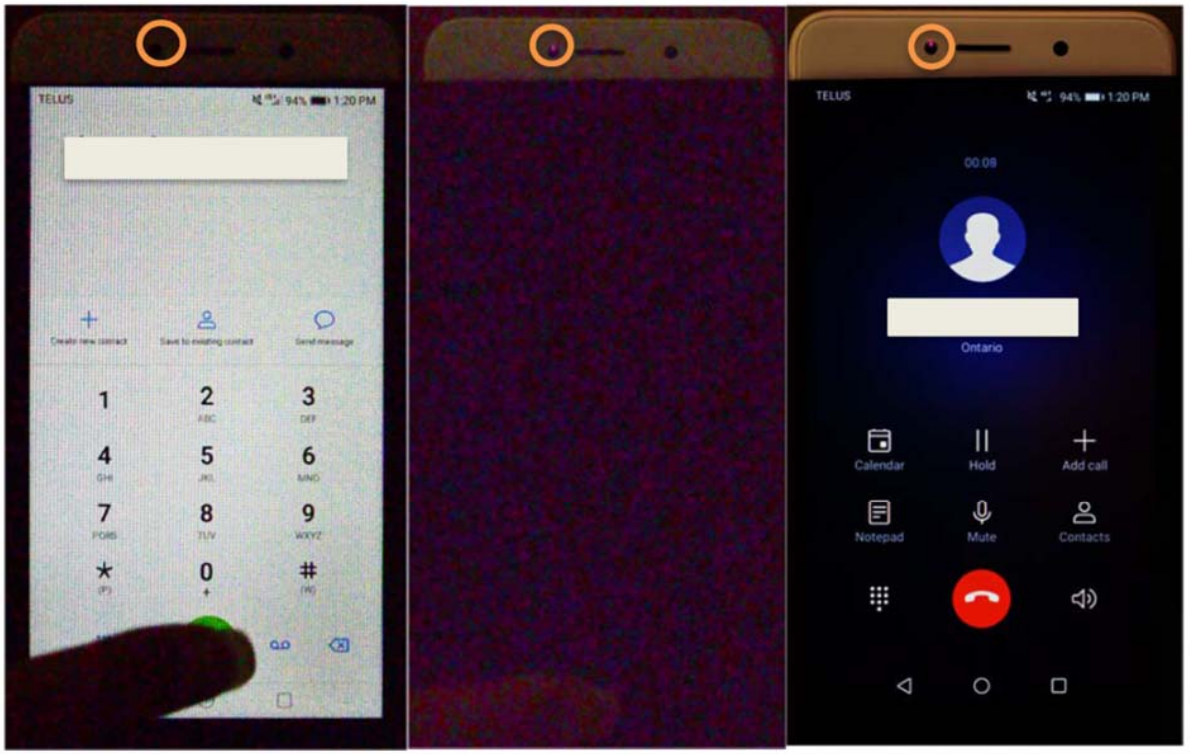
See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

140	``` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.* The Android Developer Code Website further explains: **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method. *See id.*

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Elate begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call. 

141

142

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554

143	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

144

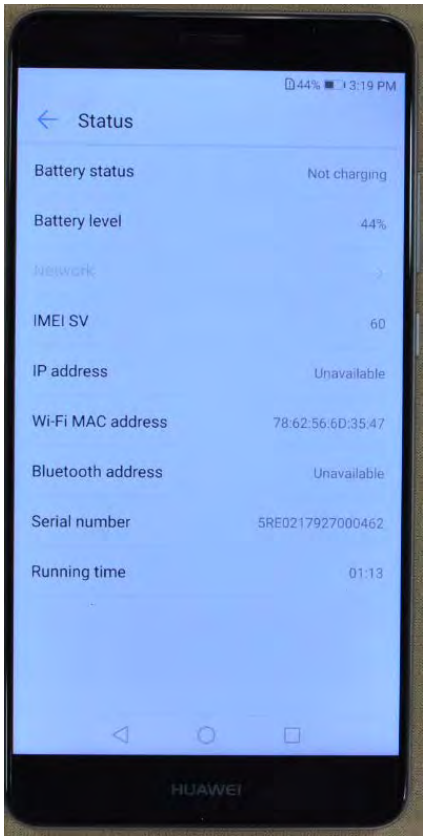
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Elate detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a

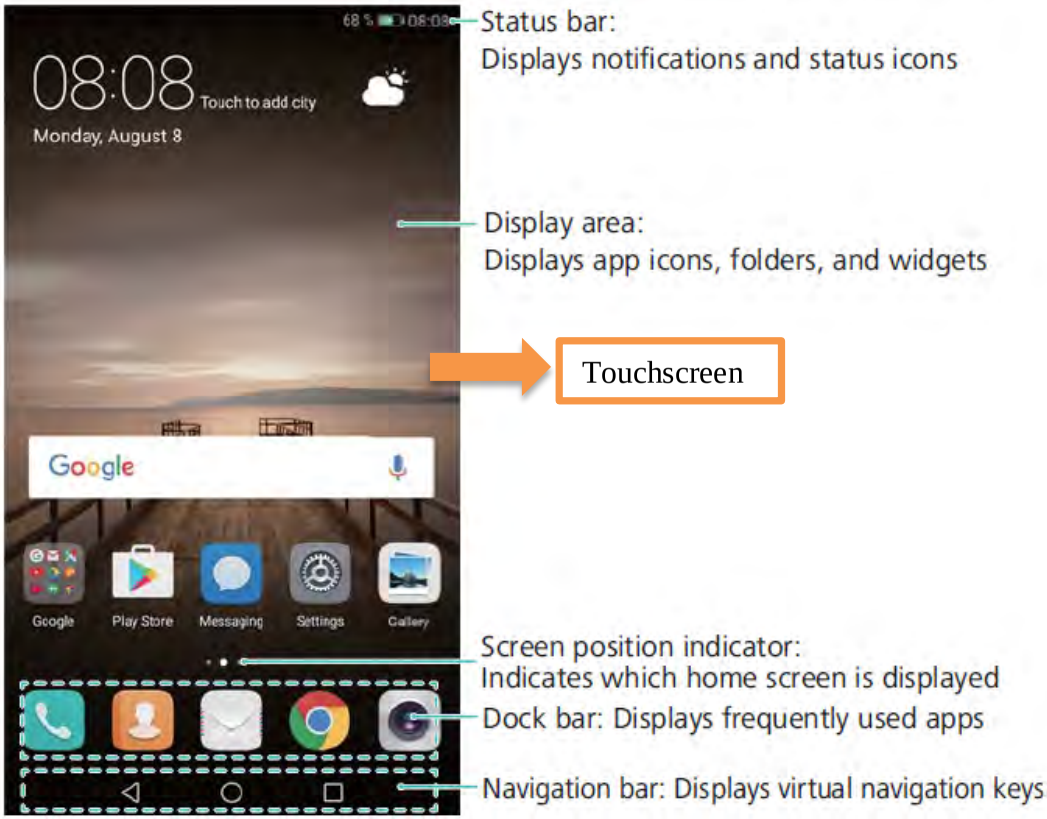
condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Elate activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Elate reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

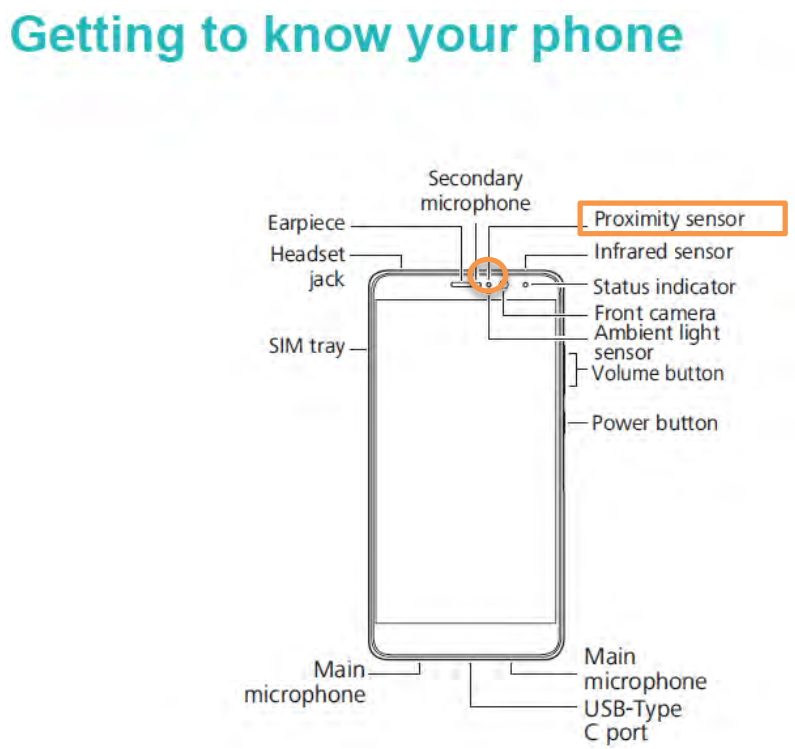
145

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Elate is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

146

Asserted Claim Elements	Huawei Mate 9
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Mate 9 is a mobile station.  

[i] a display;	The Huawei Mate 9 includes a display.  “ Use simple touchscreen gestures to perform a variety of tasks,…” *See* *http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784, last accessed Nov.19, 2018.*

[ii] a proximity sensor	The Huawei Mate 9 includes a proximity sensor.  *See* *http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784 p. 2, last accessed Nov. 19, 2018.*
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Mate 9 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Mate 9 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by

Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

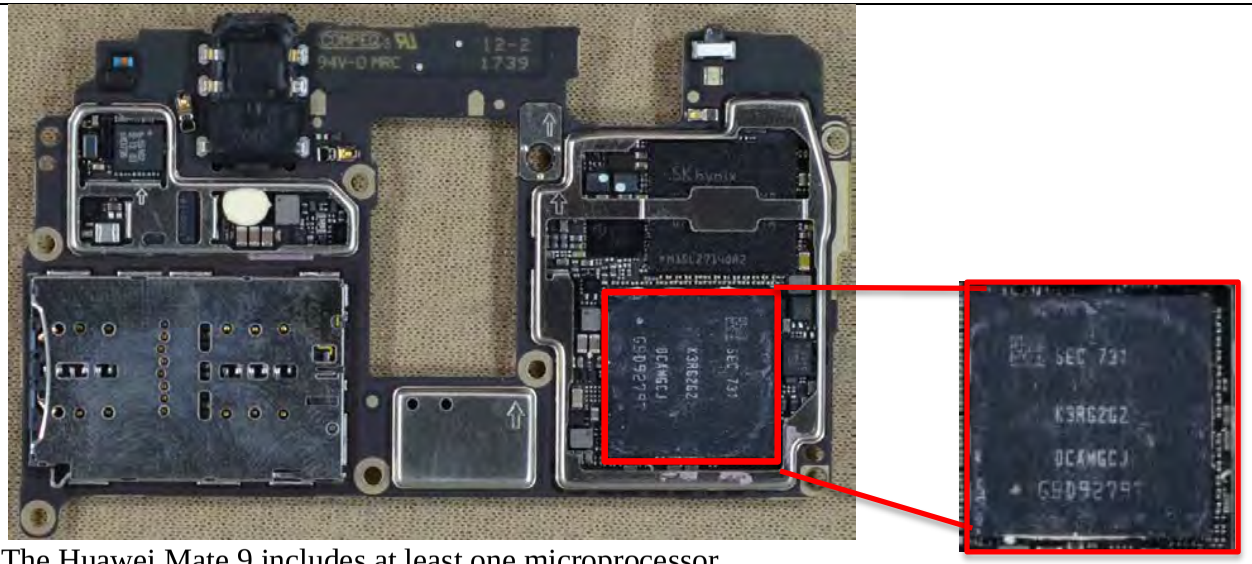
The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

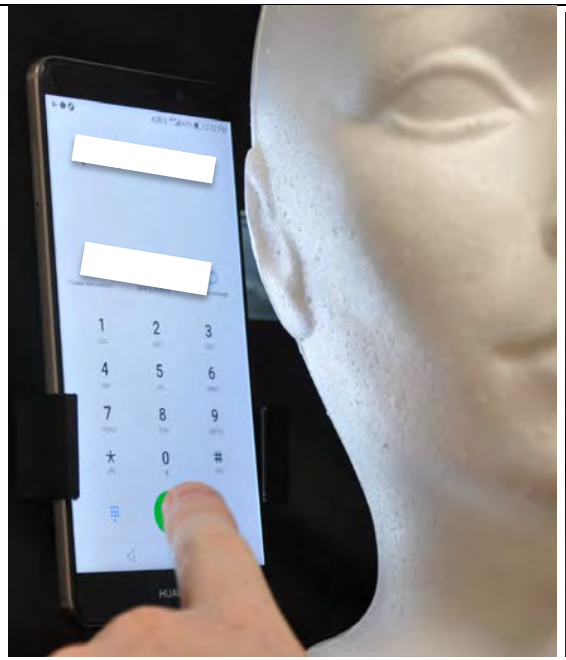
KOTLIN JAVA

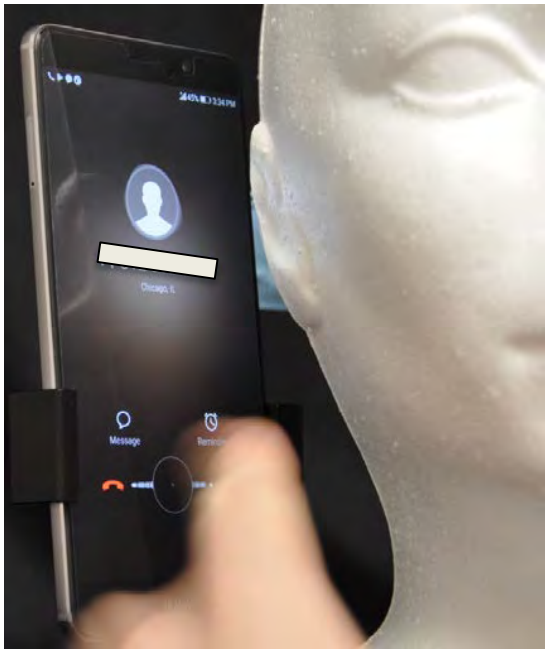
```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

```kotlin
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

[iv] a microprocessor adapted to:	 The Huawei Mate 9 includes at least one microprocessor. "**Chipset:** Huawei Kirin 960" *See https://consumer.huawei.com/us/phones/mate9/specs/, last accessed Nov. 19, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate 9 is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.





By way of example only, the Huawei Mate 9 is an Android phone. The Huawei Mate 9's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

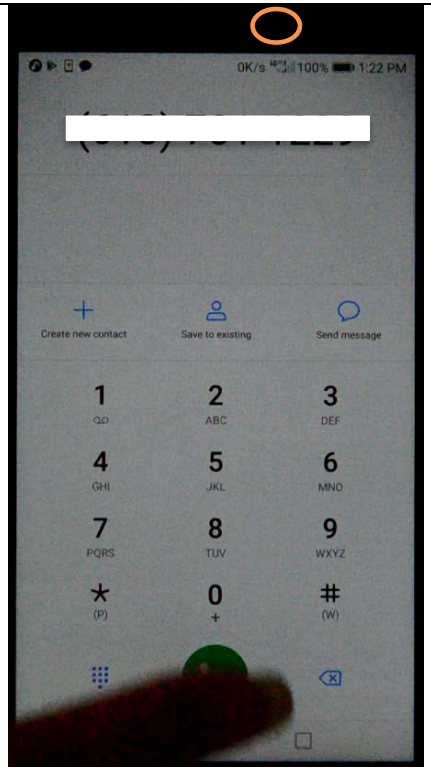
In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

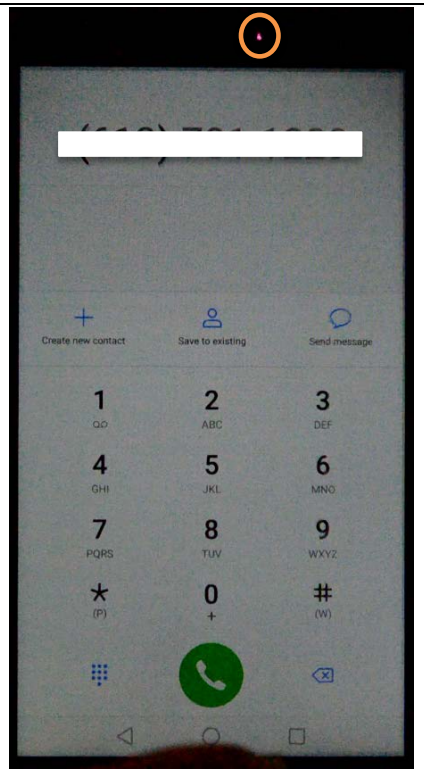
Incoming Call:

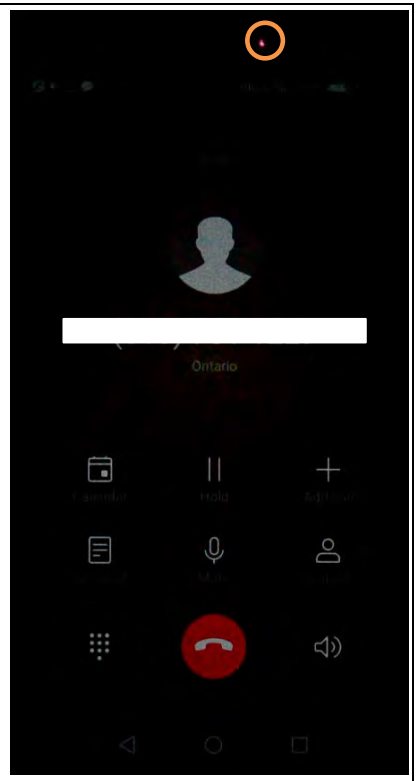
```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Mate 9 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.







By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

	See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018. In the Huawei Mate 9, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Mate 9 is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

By way of further example only, the Android Developer Code provides files that describe the proximity sensor’s signaling to the microprocessor:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

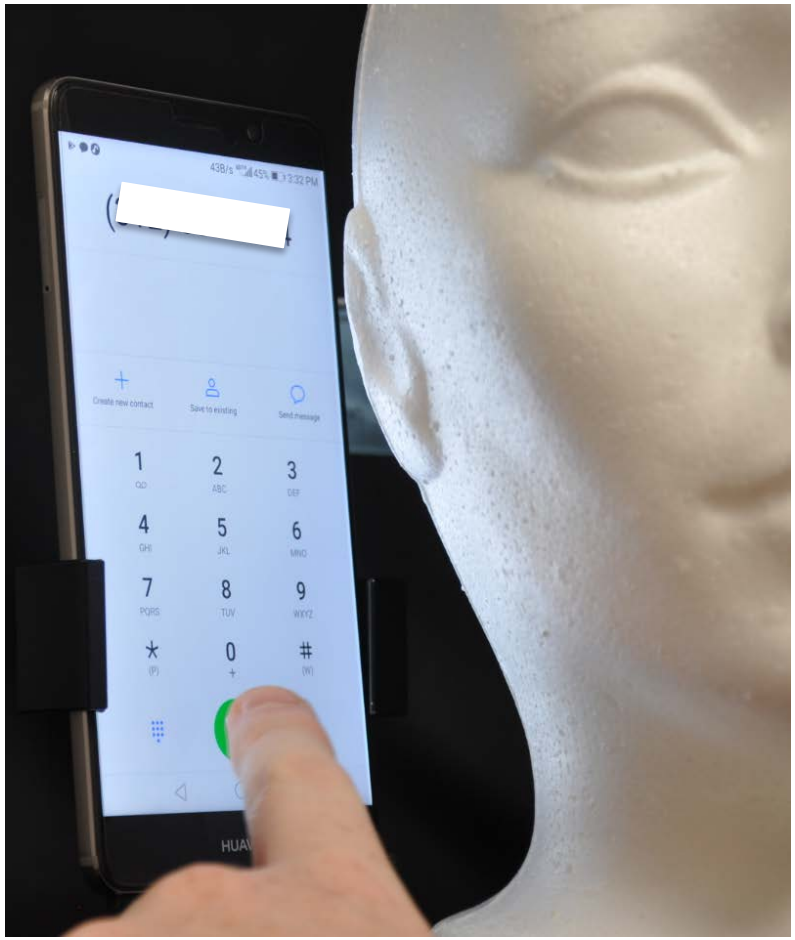
```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

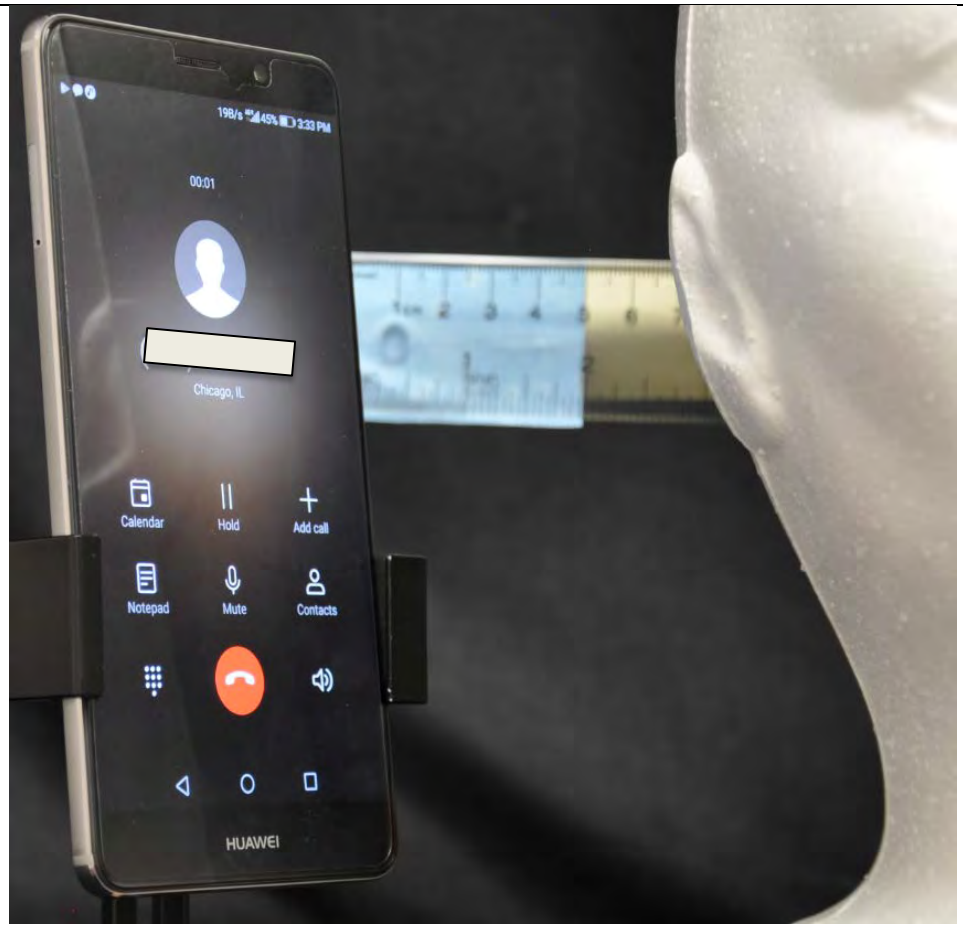
Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

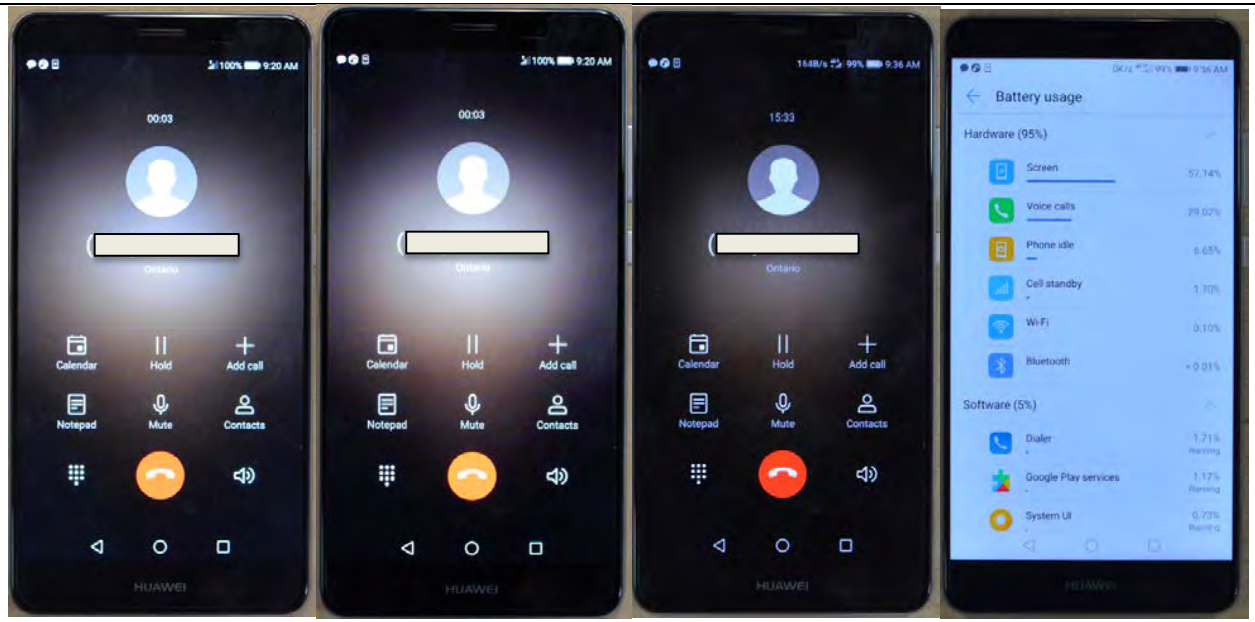
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Mate 9 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Mate 9 is proximate to an object, but no call has been initiated or answered, so the display is powered normally. 

	The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is darkened.  The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



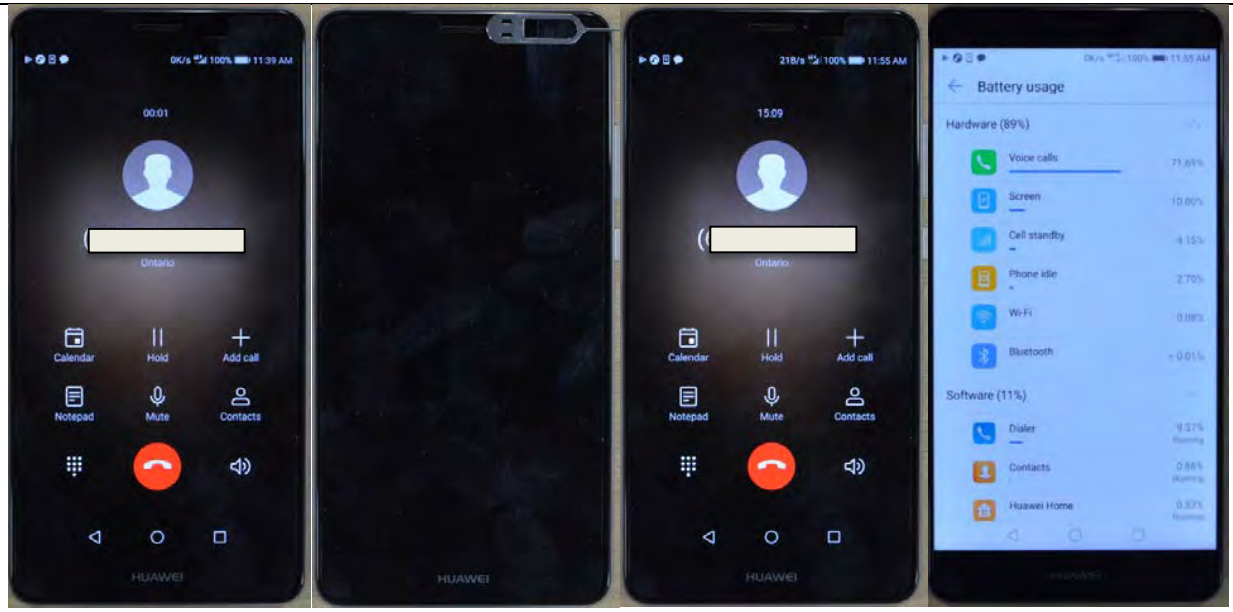
The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.14% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 10.00% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

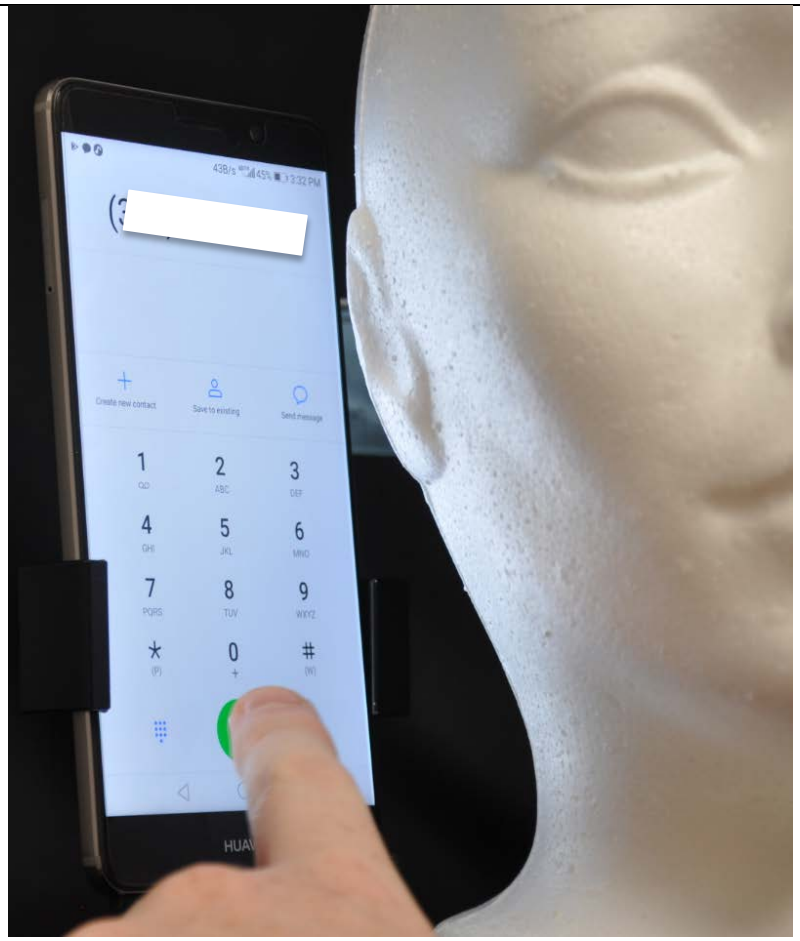
	See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018. In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)." Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

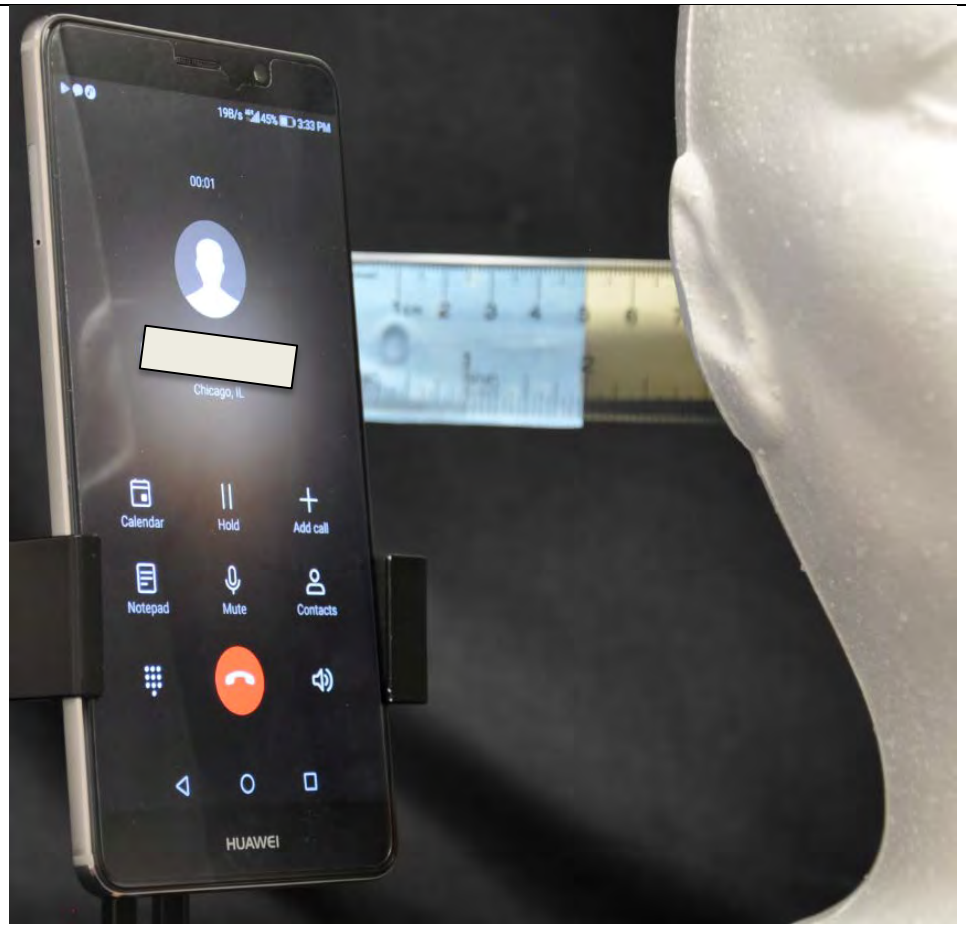
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018*
2. The mobile station of claim 1,	*See* claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Mate 9 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Mate 9 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



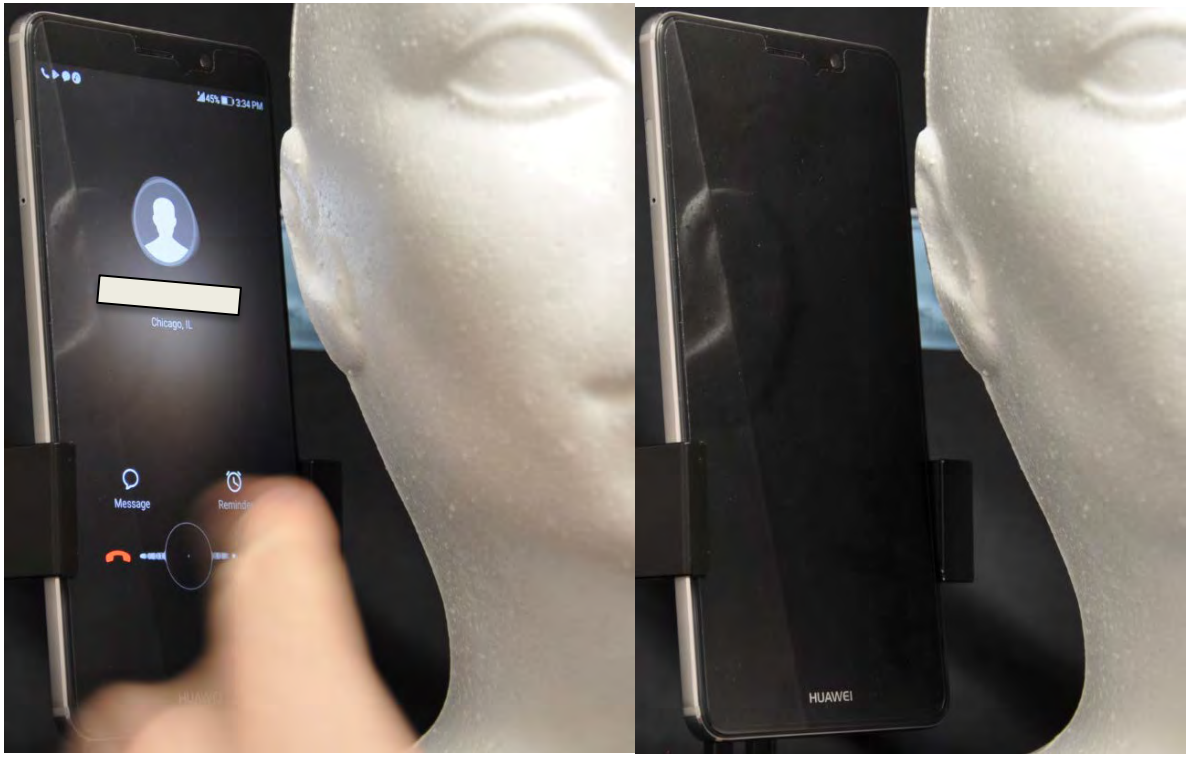
The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is darkened.

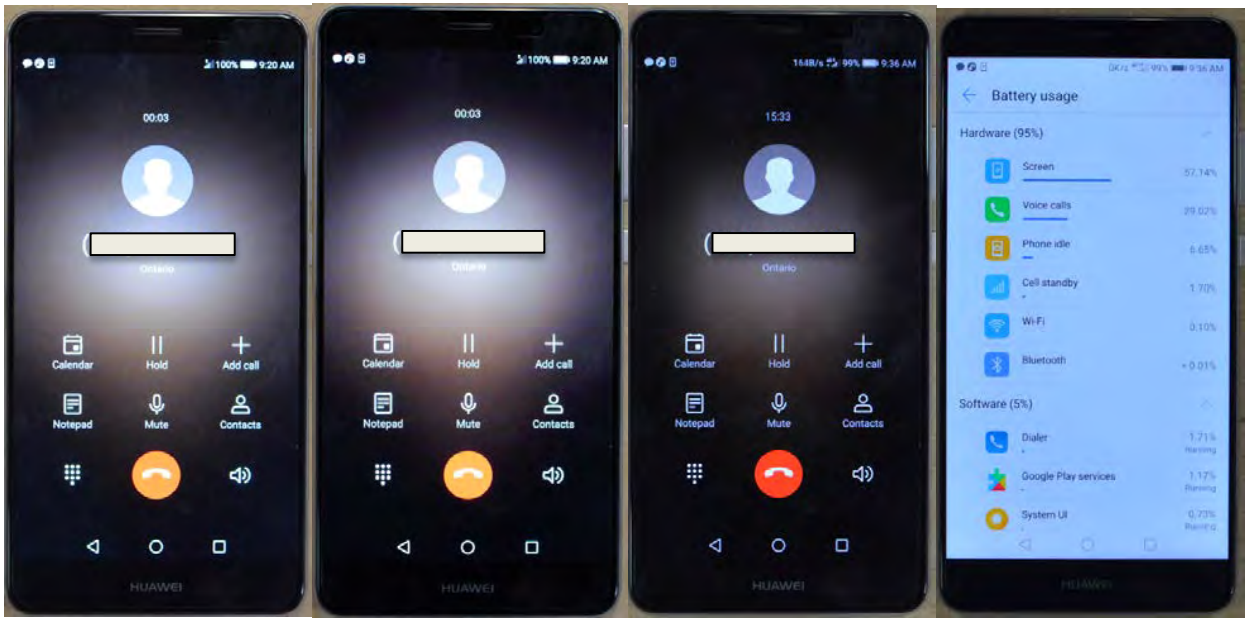
The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

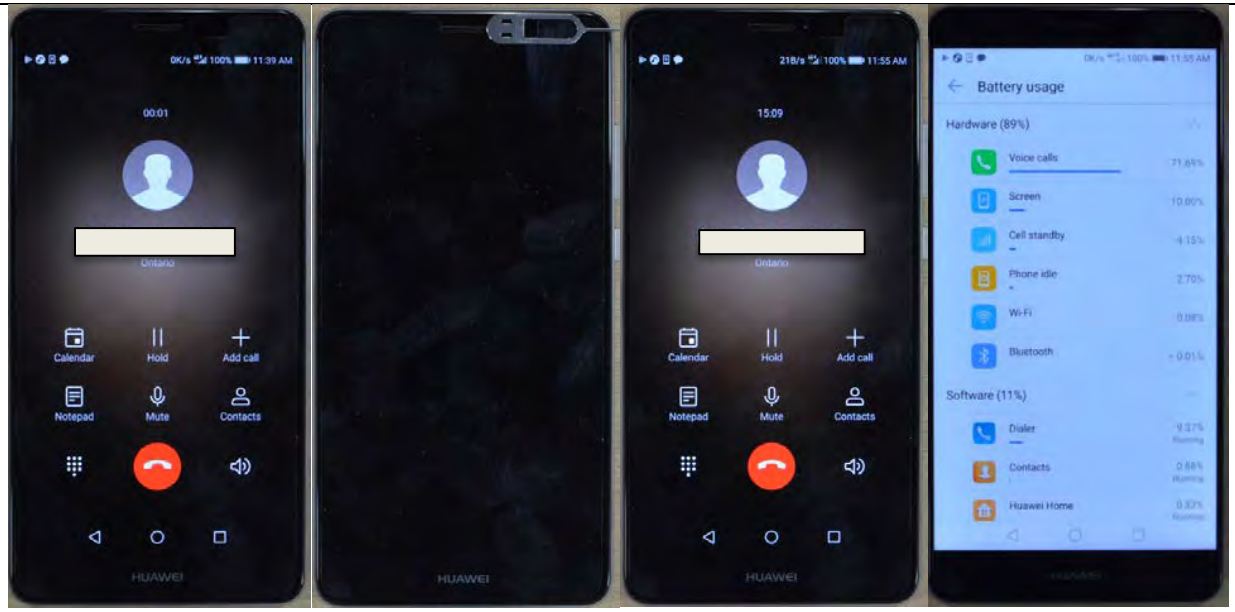


By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)."

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 30, 2018.

4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Mate 9 reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

	By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally  When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.14% of the battery usage. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 10.00% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

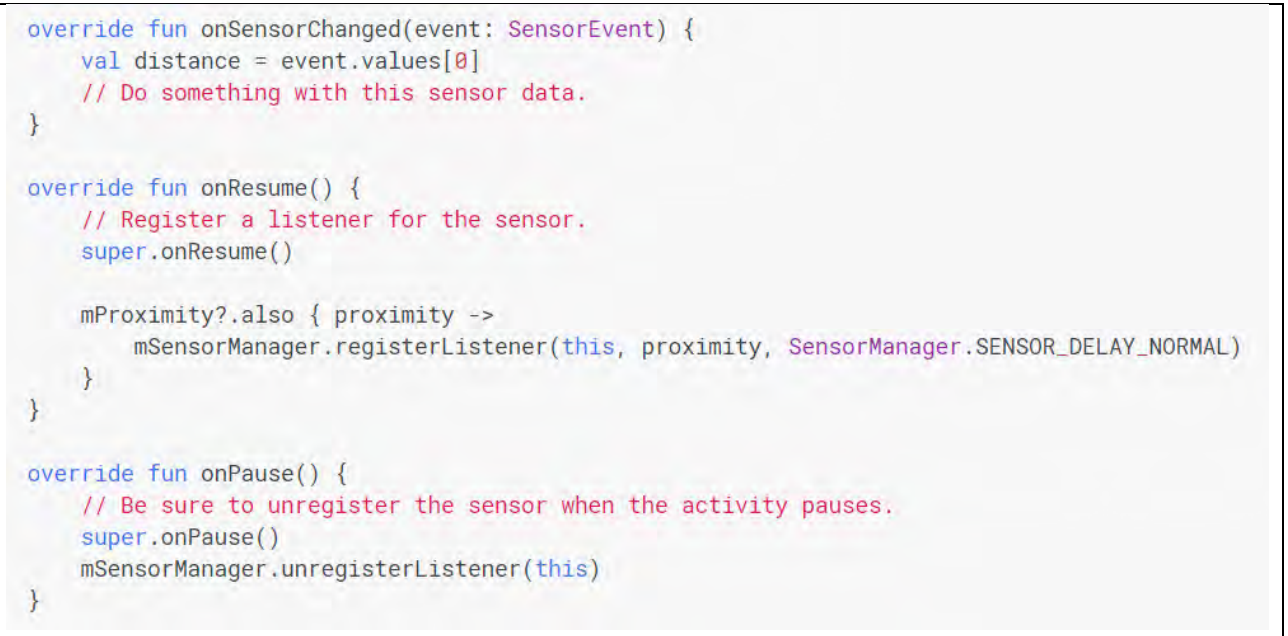
KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

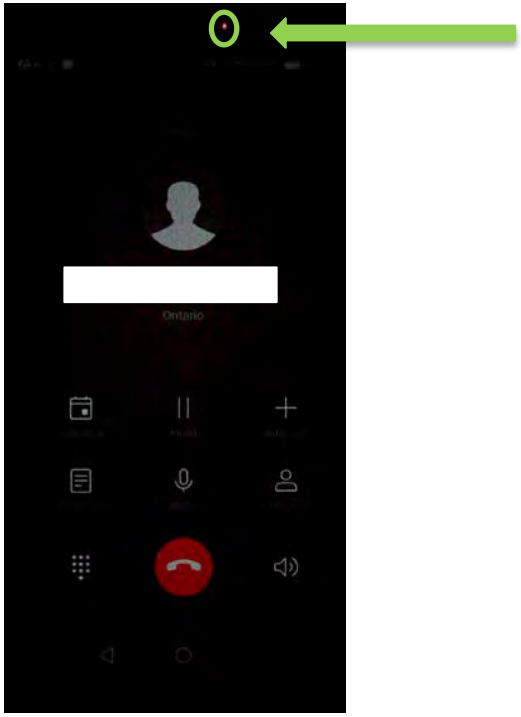
The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

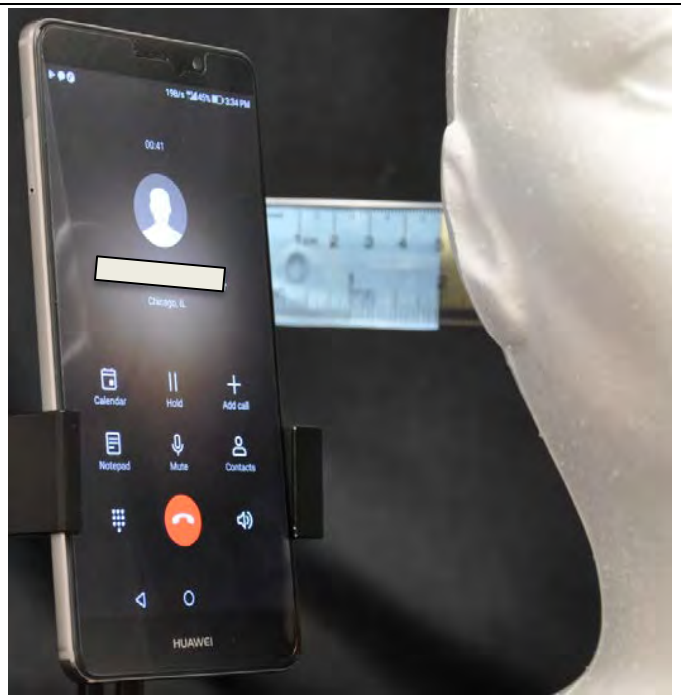
	 *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Mate 9 is a range-detecting sensor.

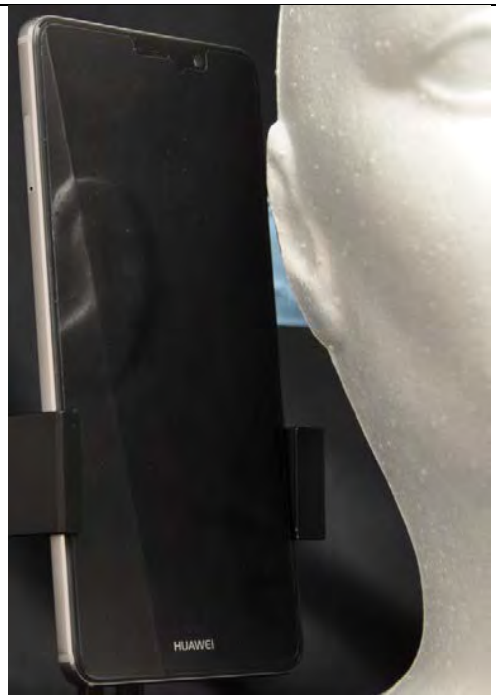
The proximity sensor detects the range of a proximate object:



The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

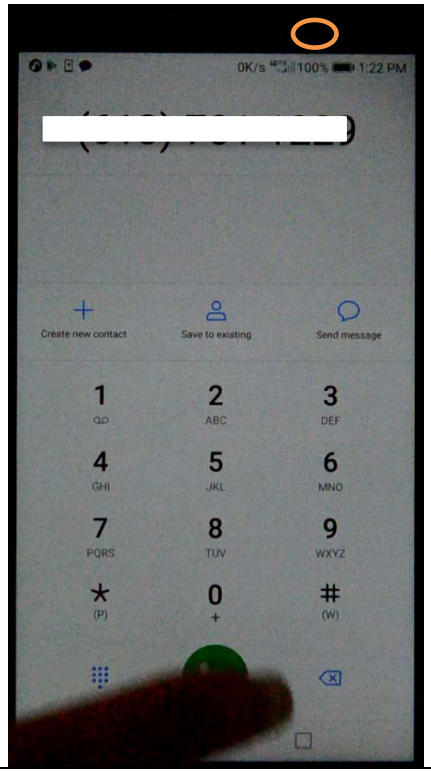
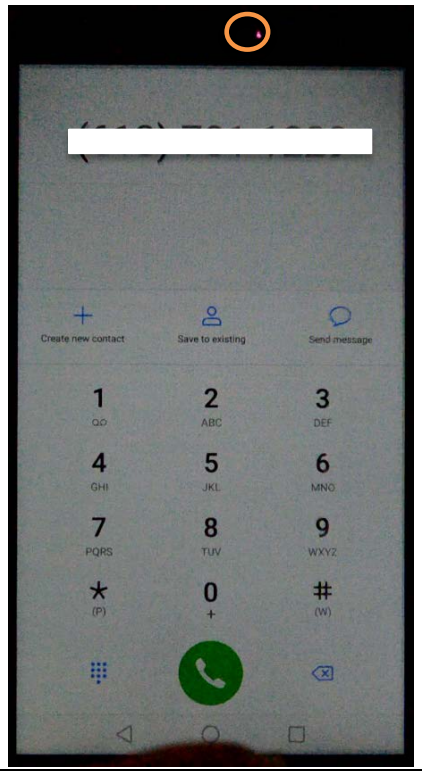
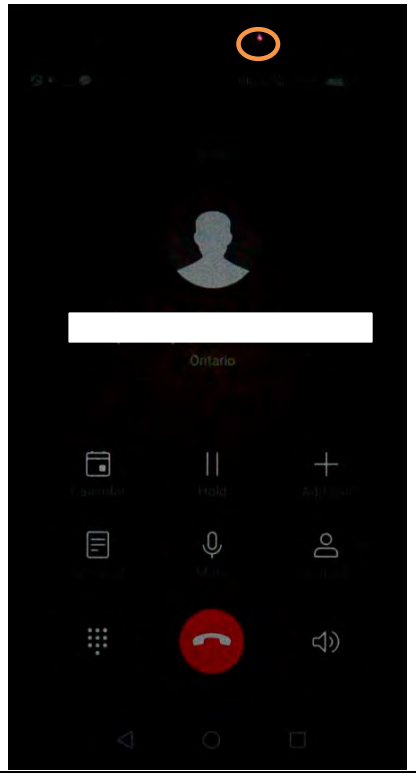
```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Mate 9 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.   

	In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different	The Huawei Mate 9 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.

from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	*See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Mate 9 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Mate 9 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].
and a microprocessor adapted to:	*See* claim[iv].

(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate 9 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

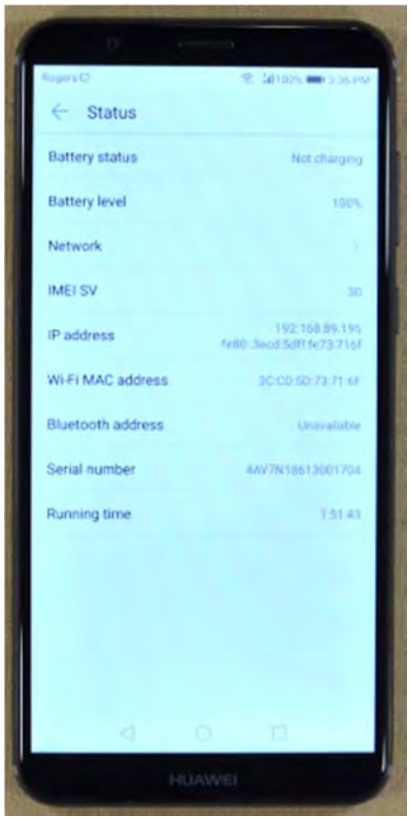
Asserted Claim Elements	Huawei Mate SE
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Mate SE is a mobile station.  

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

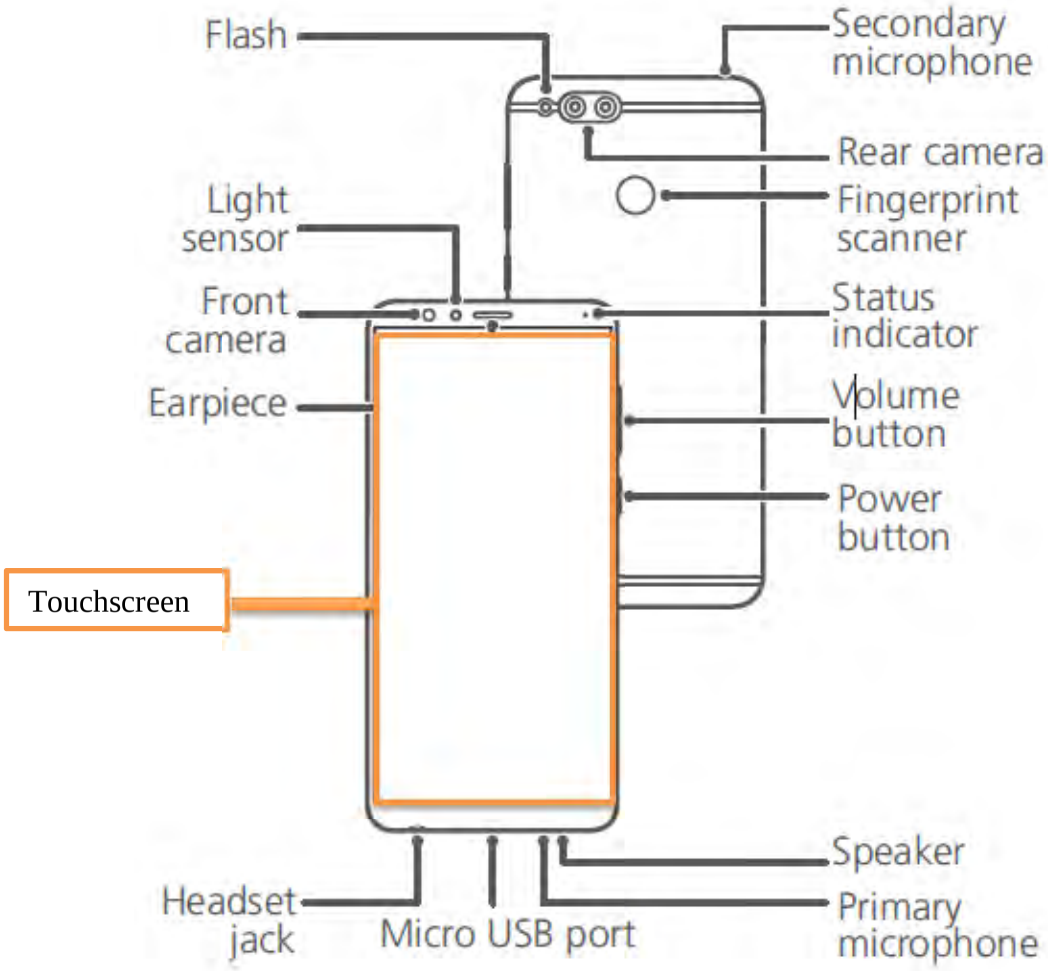
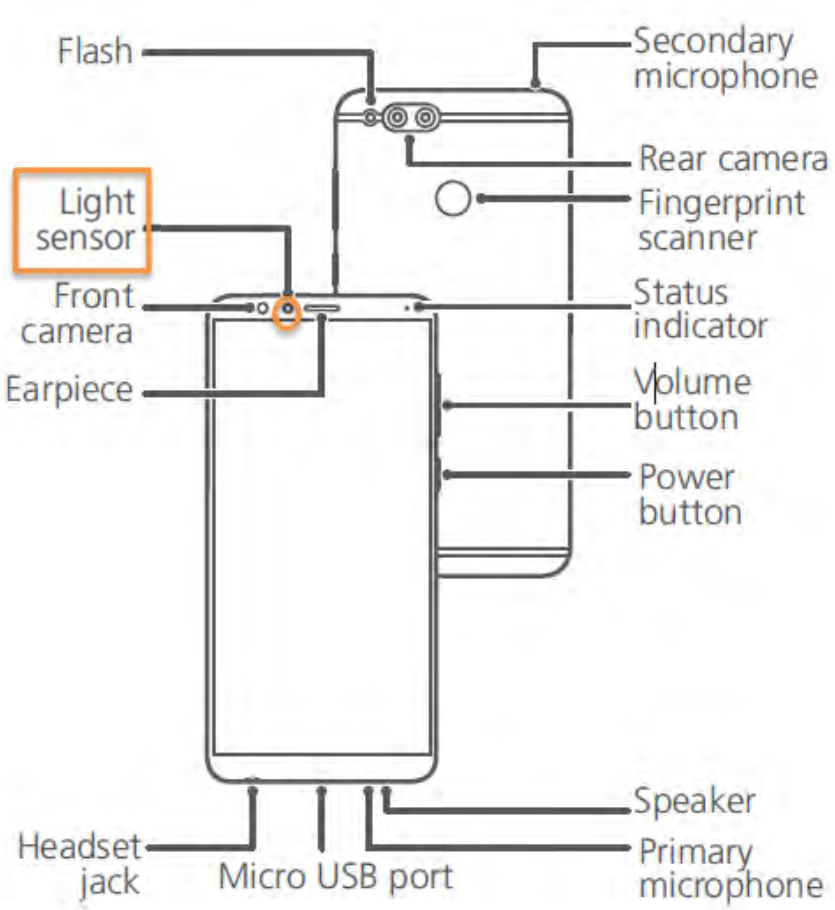
[i] a display;	The Huawei Mate SE includes a display.  *See* *http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632 p. 1, last accessed Nov 22, 2018.*

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Mate SE includes a proximity sensor.  *See http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632, last accessed Nov 22, 2018.*

196

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Mate SE is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Mate SE is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY — Hardware — Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. — Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

198

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

199

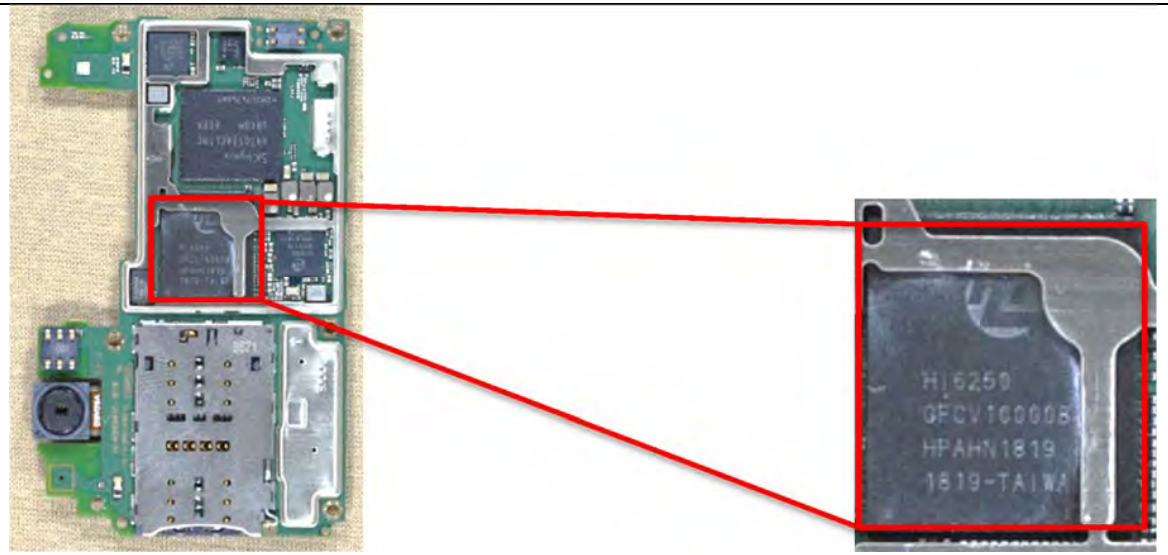
Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

200	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

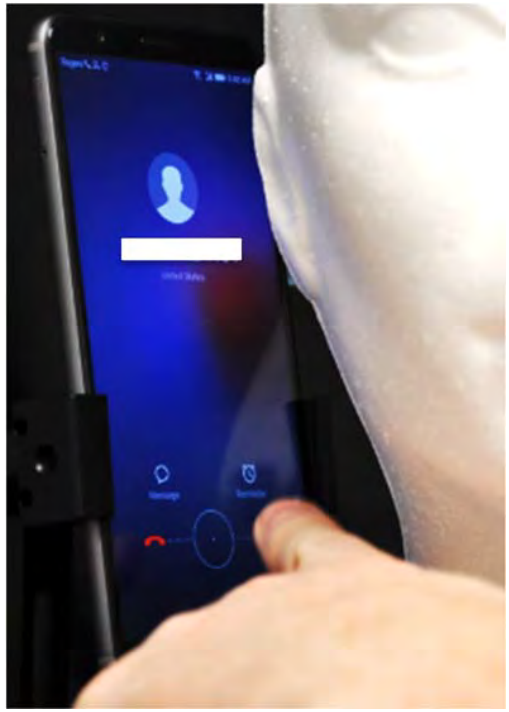
Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Mate SE includes at least one microprocessor.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	 "**Chipset**: Kirin 659, Octa-Core (4 x 2.36 GHz+4 x 1.7 GHz)" *See* *https://consumer.huawei.com/us/phones/mate-se/specs/* (last accessed Nov 22, 2018)
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate SE is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.

202

203	  By way of example only, the Huawei Mate SE is an Android phone. The Huawei Mate SE's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call:

204

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

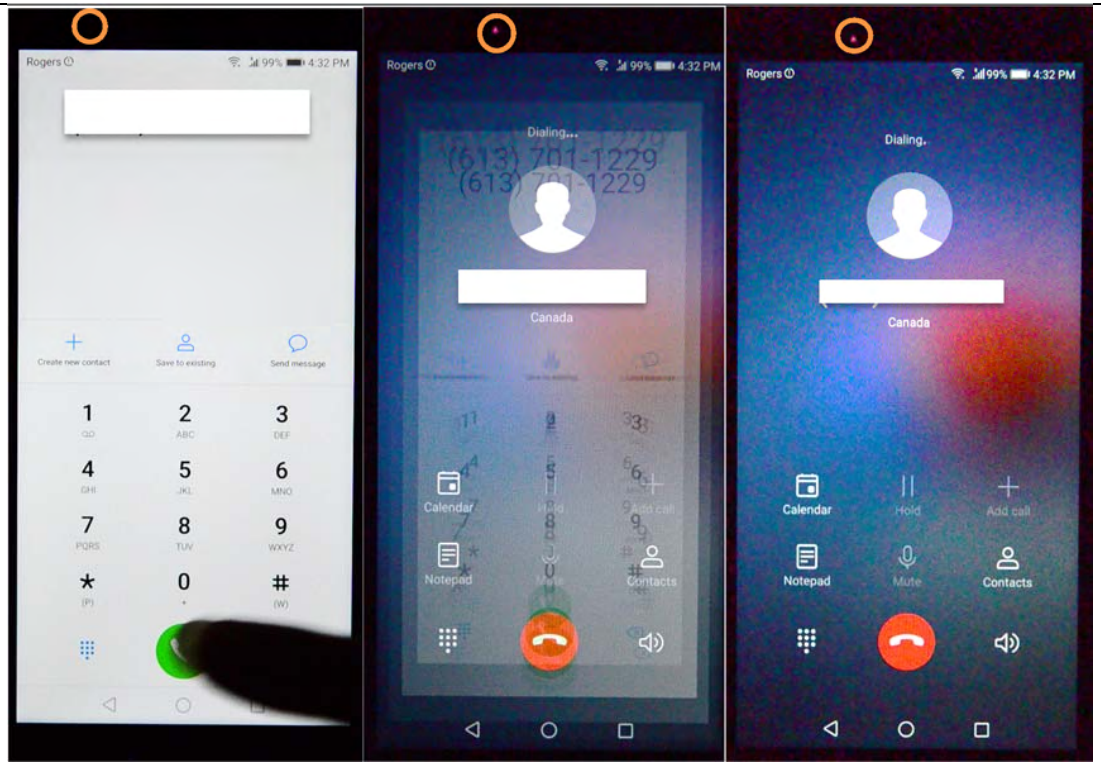
```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Mate SE is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

205

206



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

207

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

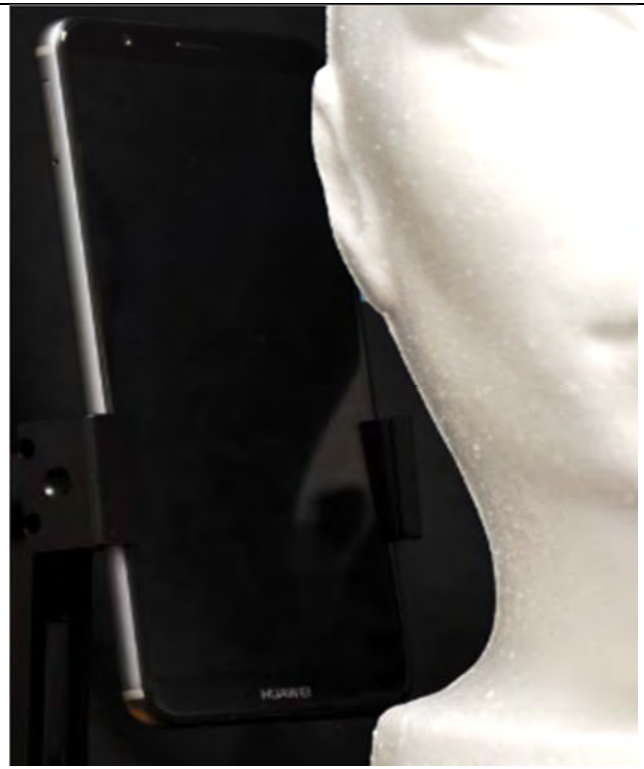
Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Mate SE, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Mate SE is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

208

	 By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

209

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

210	TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

211

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Mate SE is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Mate SE is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is darkened.

212

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

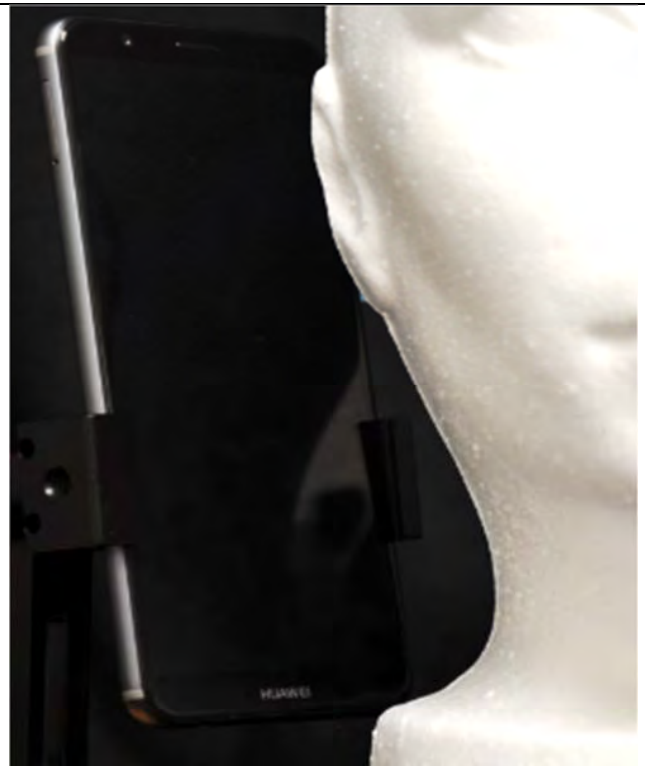
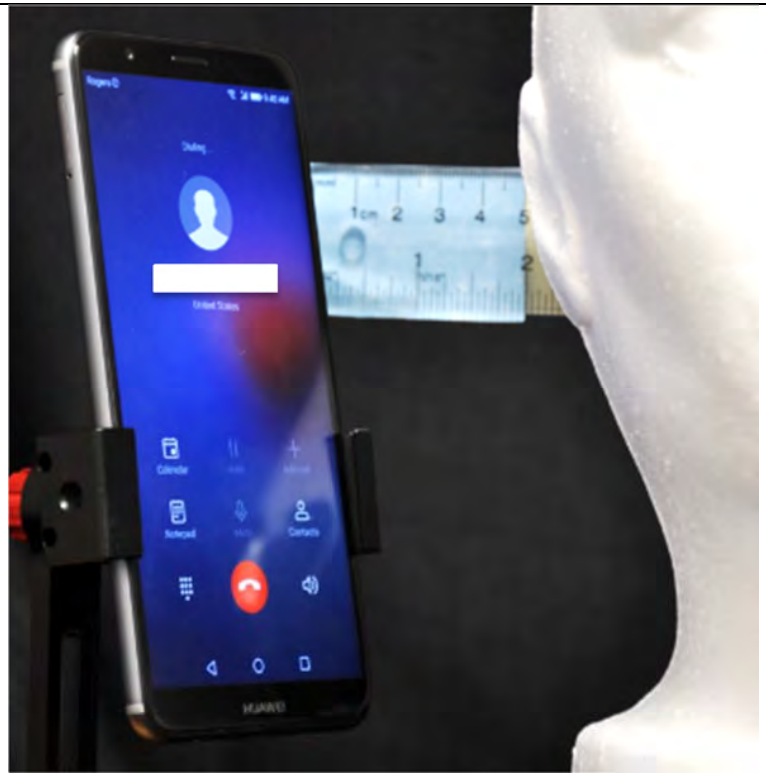
213	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

214



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

215	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 99%. The screen consumed 72.62% of the battery. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 20.13% of the battery.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

217	*See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)." Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

218	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018*
2. The mobile station of claim 1,	*See* claim 1.

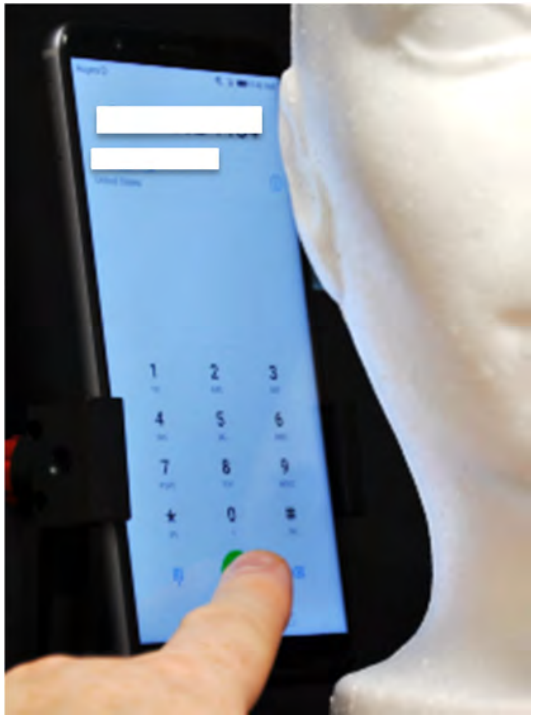
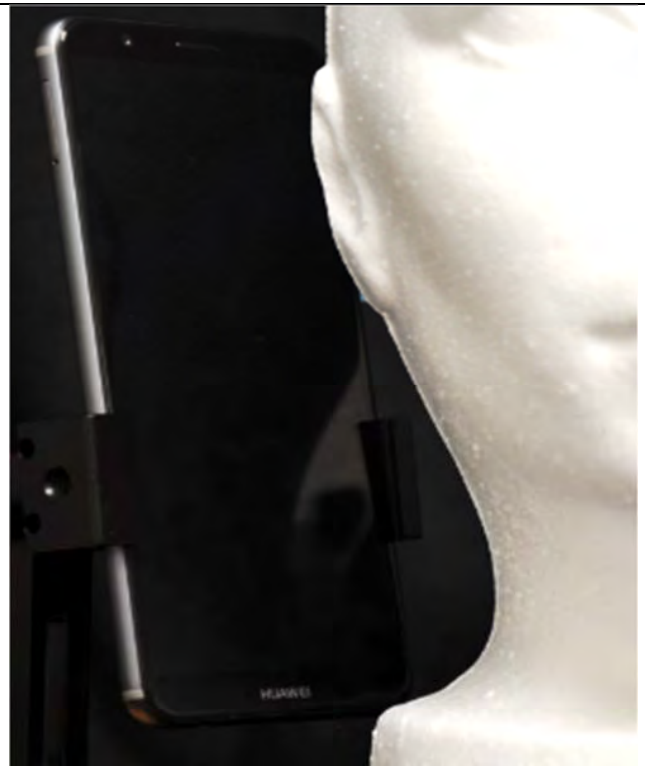
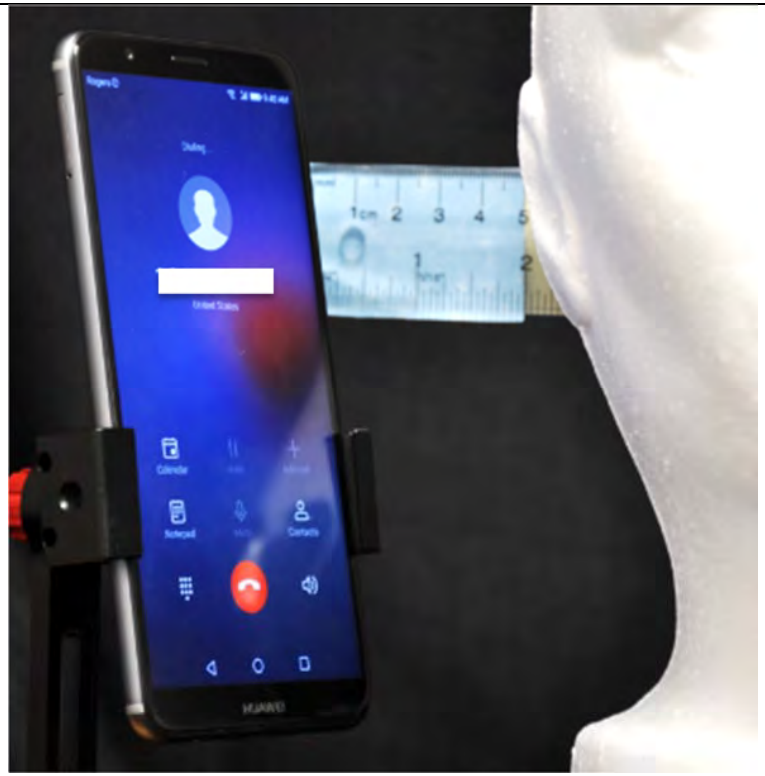
further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The microprocessor of the Huawei Mate SE increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Mate SE is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is darkened.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

221

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

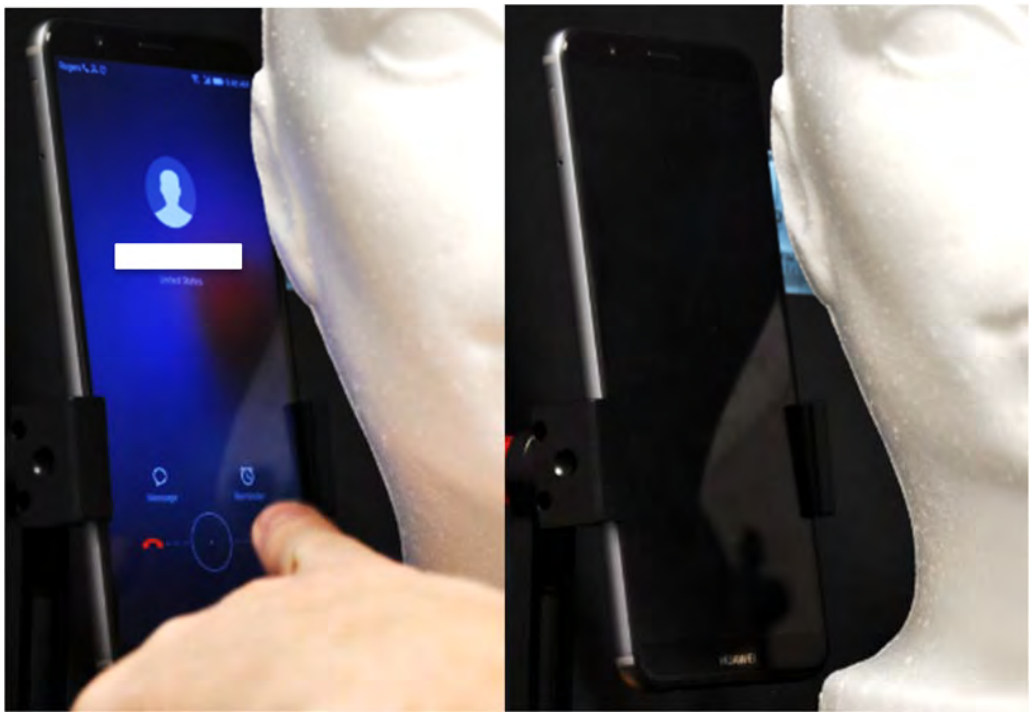
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Mate SE reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 99%. The screen consumed 72.62% of the battery.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

225

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 20.13% of the battery.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

226

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

227

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

228

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

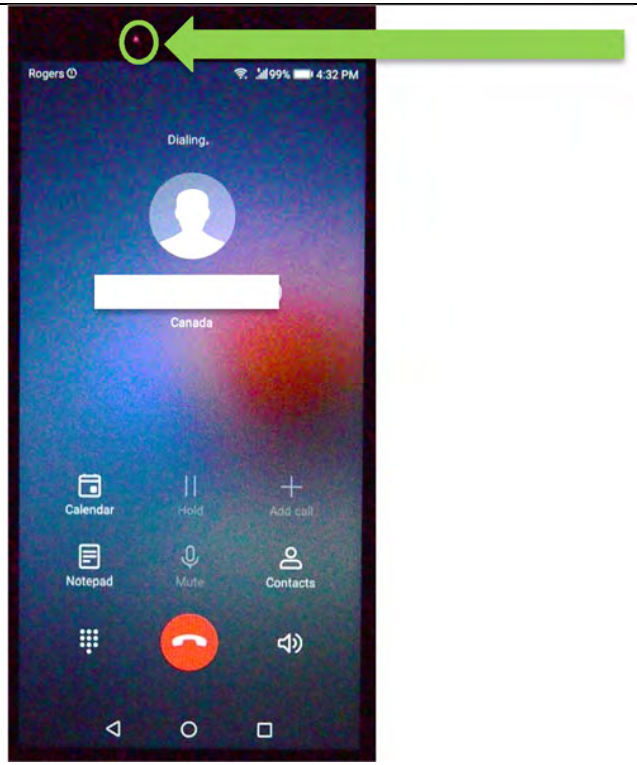
229	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Mate SE is a range-detecting sensor.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

231

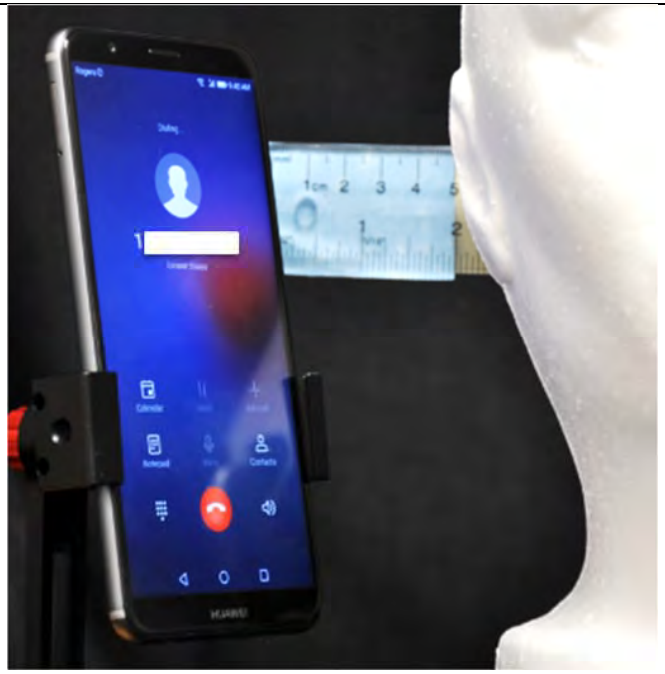


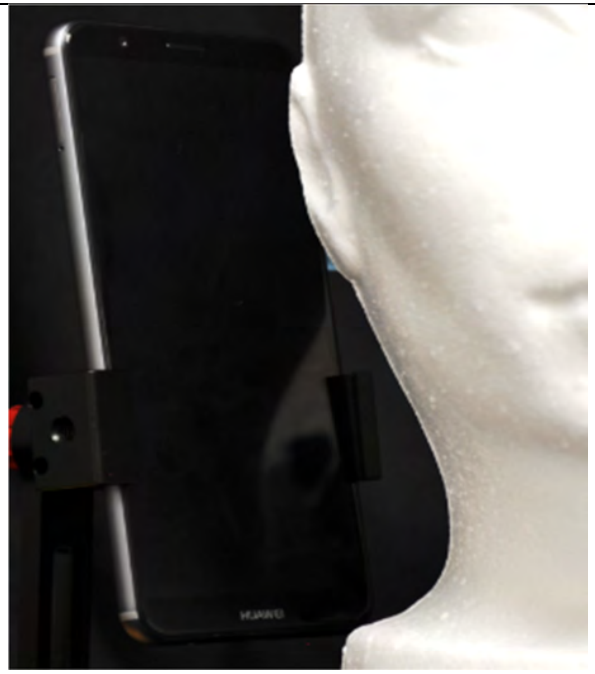
The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

The proximity sensor detects the range of a proximate object:

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.

232





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

233

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

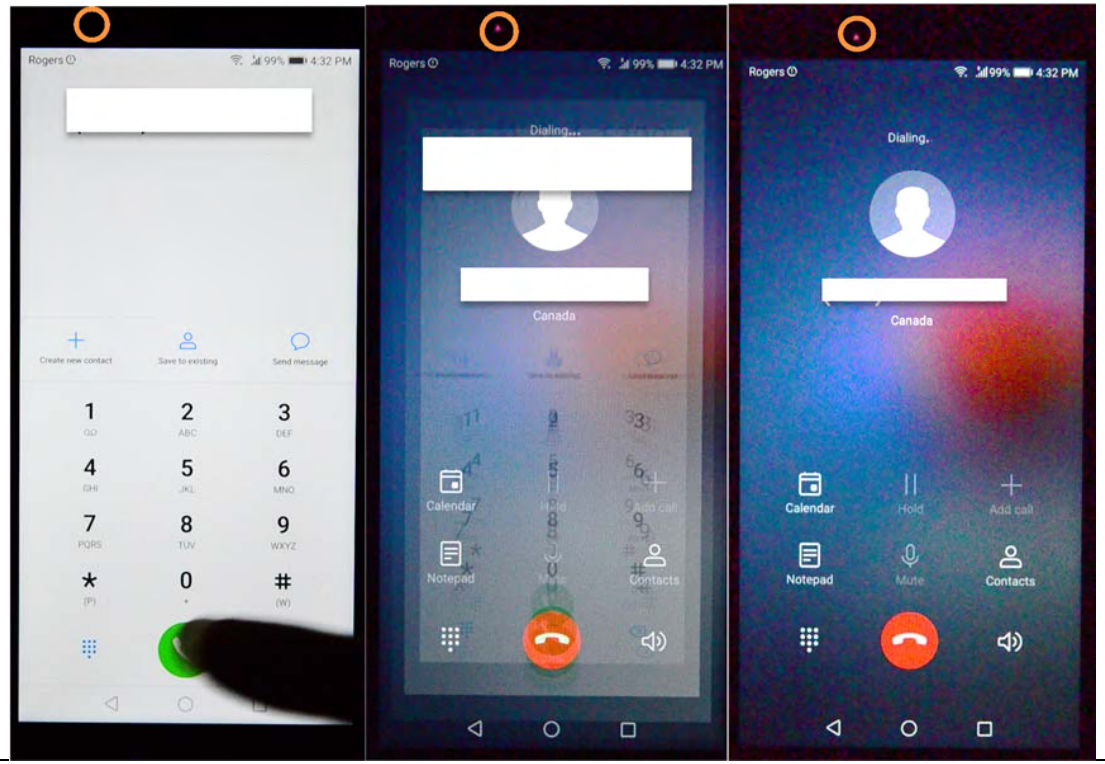
7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Mate SE begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call. 

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

235	In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554

236	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

237

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Mate SE detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Mate SE activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Mate SE reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Mate SE is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

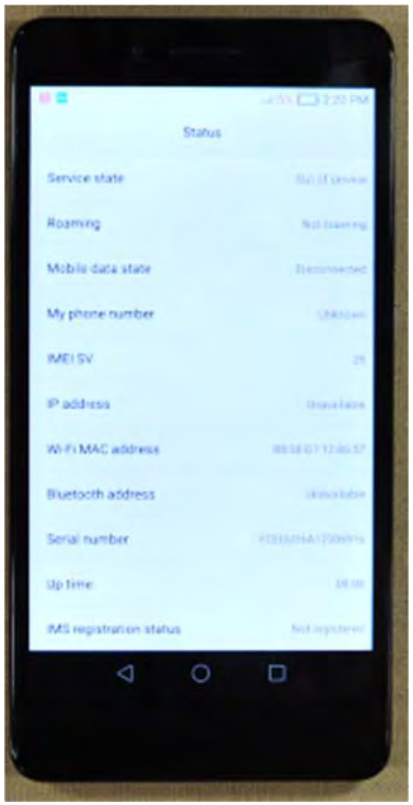
Asserted Claim Elements	Huawei Sensa
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Sensa is a mobile station.  

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

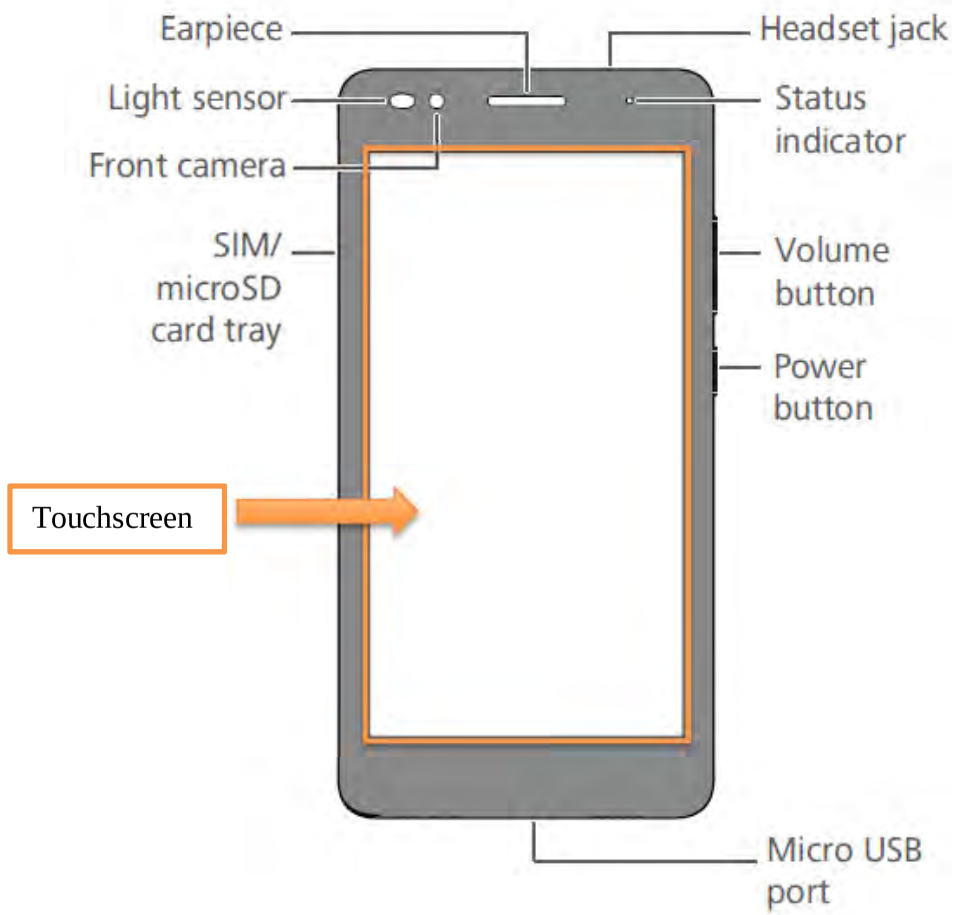
[i] a display;	The Huawei Sensa includes a display.  "Use simple **touchscreen** gestures to perform a variety of tasks, such as open applications, scroll through lists, and zoom images."

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

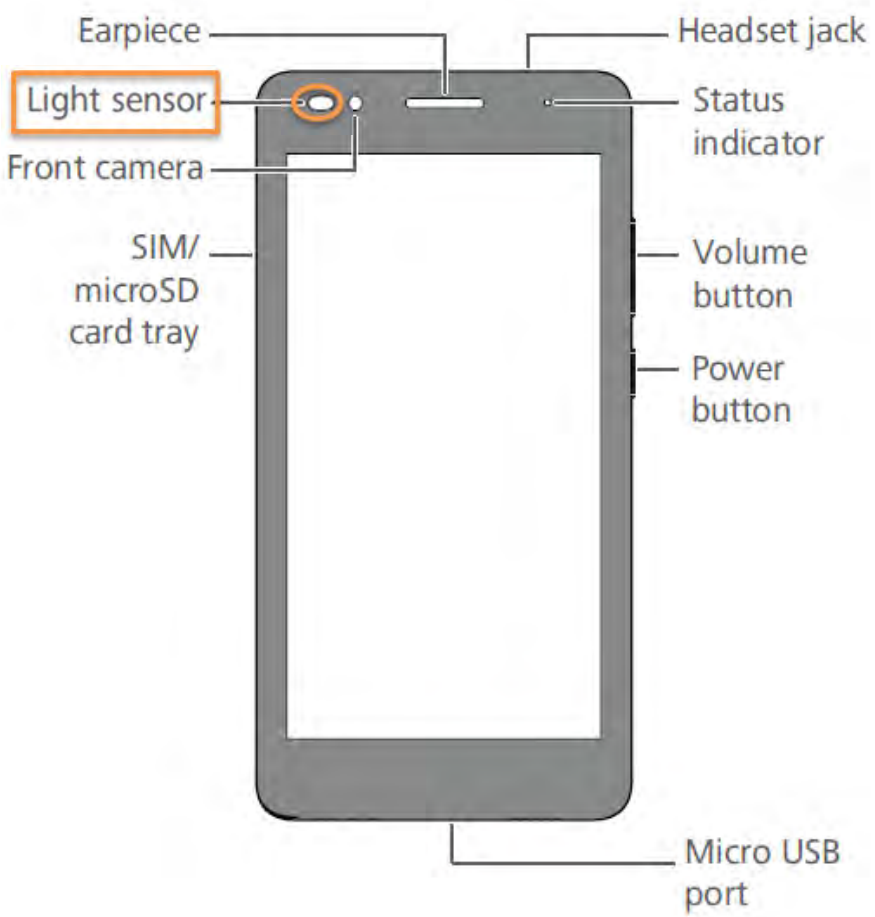
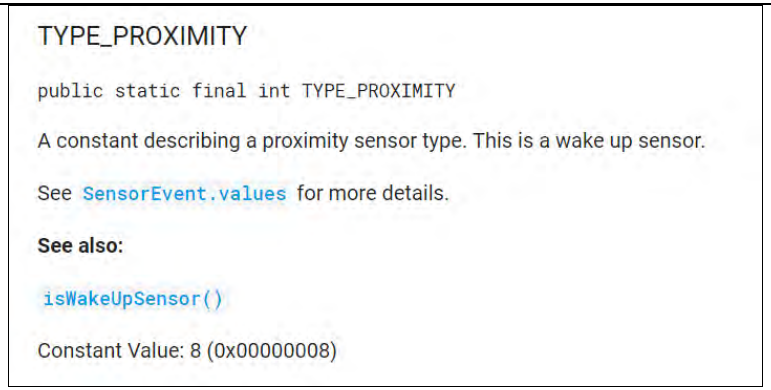
	See http://consumer-tkb.Huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812 pp. 1 & 8, last accessed Nov 23, 2018.
[ii] a proximity sensor	The Huawei Sensa includes a proximity sensor. 

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	See http://consumer-tkb.Huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812 p. 1, last accessed Nov 23, 2018.
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Sensa is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Sensa is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

243



TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

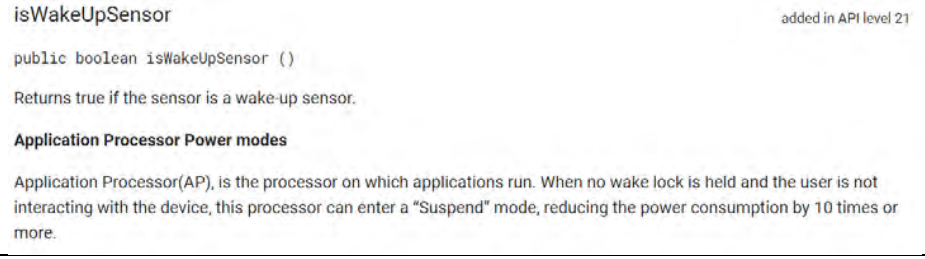
See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:



isWakeUpSensor

added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

244

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

245	**Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

247

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

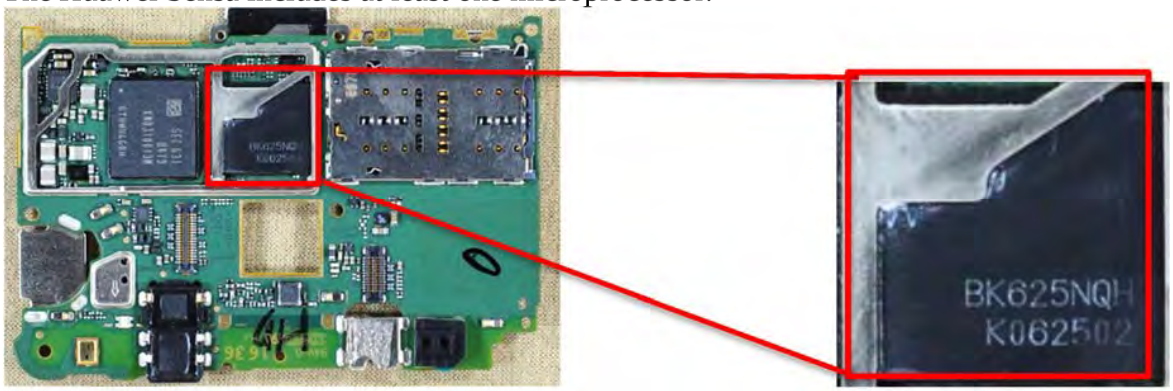
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Sensa includes at least one microprocessor. 

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

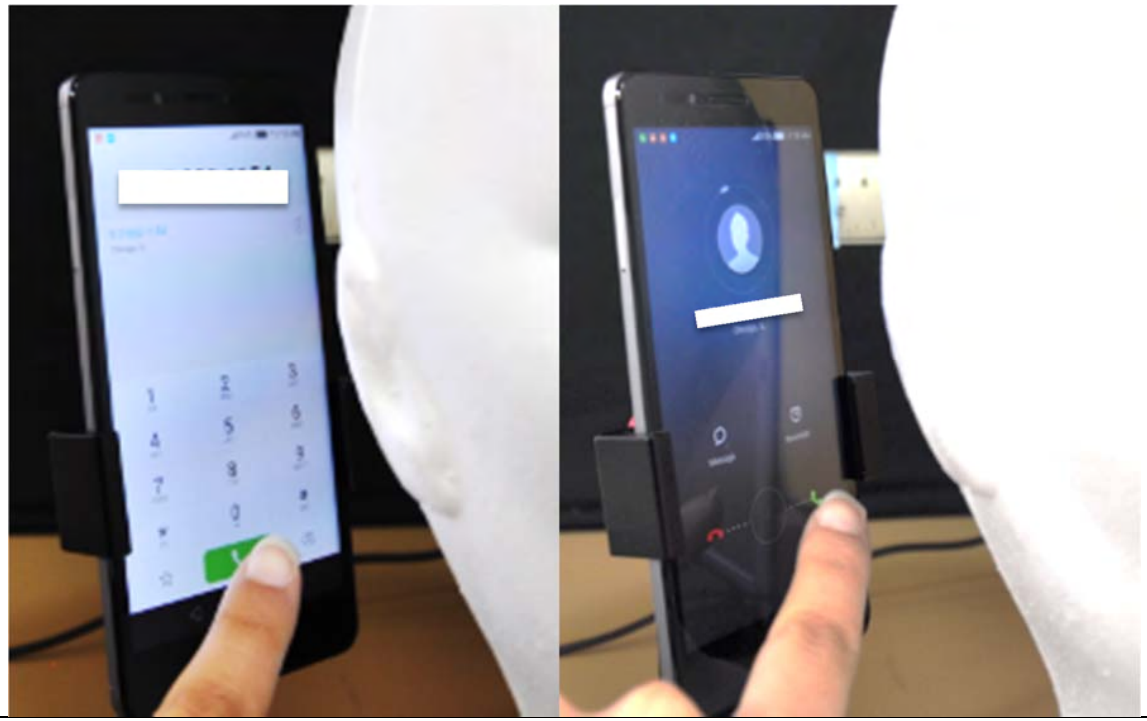
	"**Chipset:** QUALCOMM " *See https://www.cnet.com/products/Huawei-sensa-lte-4g-lte-16-gb-gsm-smartphone/specs/, last accessed Nov 23, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Sensa is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call. 

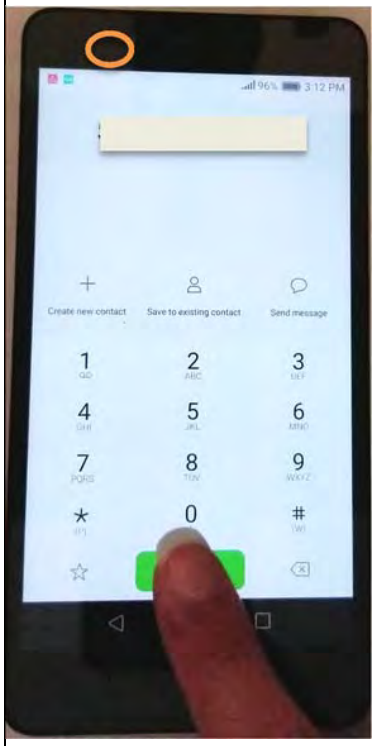
Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

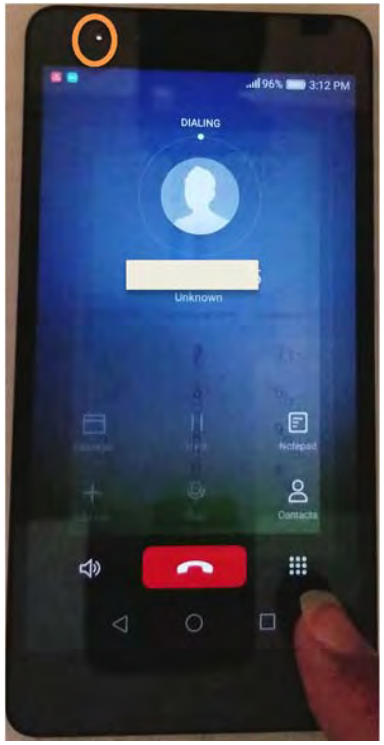
250	By way of example only, the Huawei Sensa is an Android phone. The Huawei Sensa's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `  public OutgoingReceiver() {` `  }` `  @Override` `  public void onReceive(Context context, Intent intent) {` `  String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `  Toast.makeText(ctx,` `  "Outgoing: "+number,` `  Toast.LENGTH_LONG).show();` `  }` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Sensa is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554





By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

252

253

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Sensa, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Sensa is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

254

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	 By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

255

256

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

257

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Sensa is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Sensa is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is darkened.

258

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

259

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

260	 The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

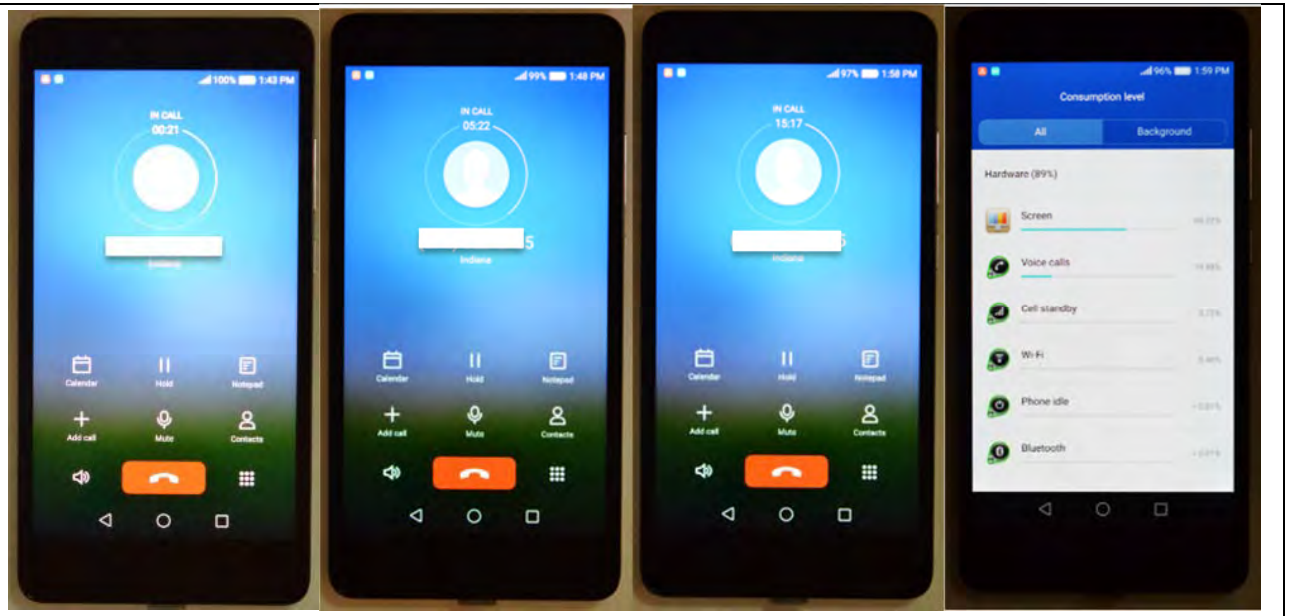
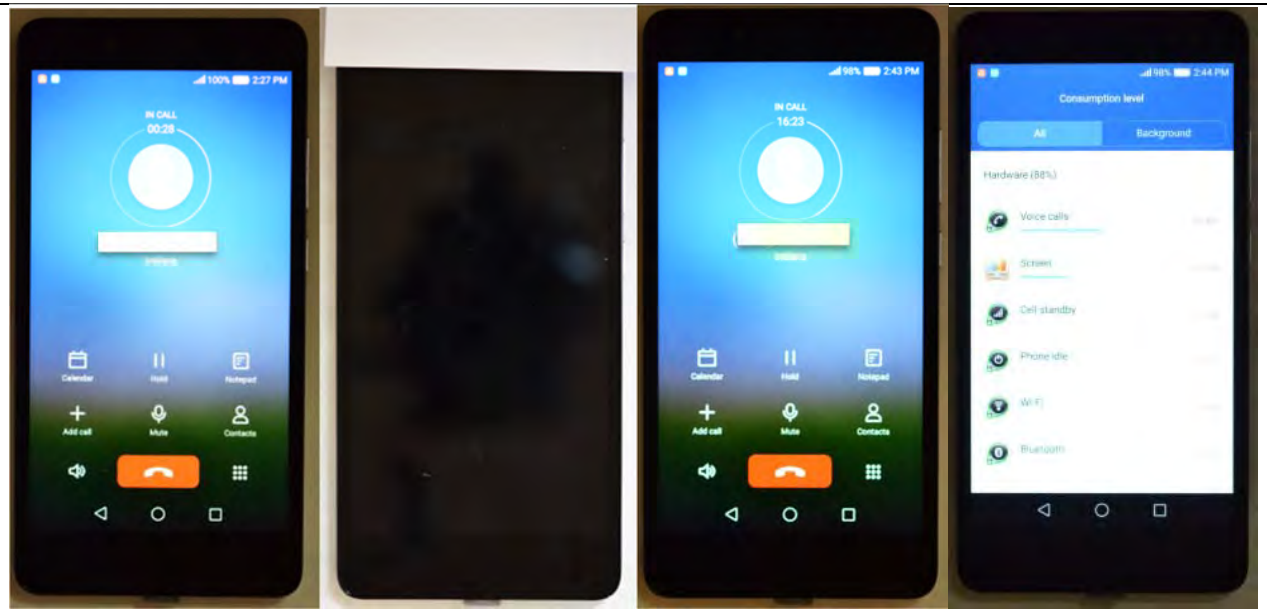
261	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 96%. The screen consumed 68.22% of the battery. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

262



When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 31.15% of the battery.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

263

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

264	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018*
2. The mobile station of claim 1,	*See* claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The microprocessor of the Huawei Sensa increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Sensa is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is darkened.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

267

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

268	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Sensa reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call. By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

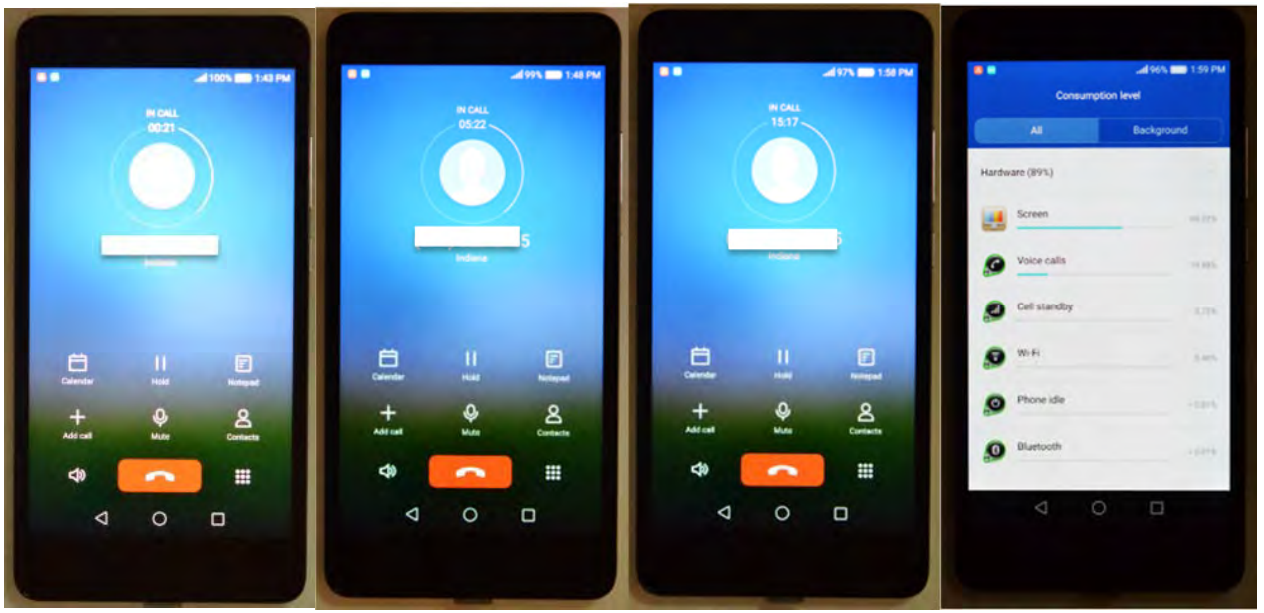
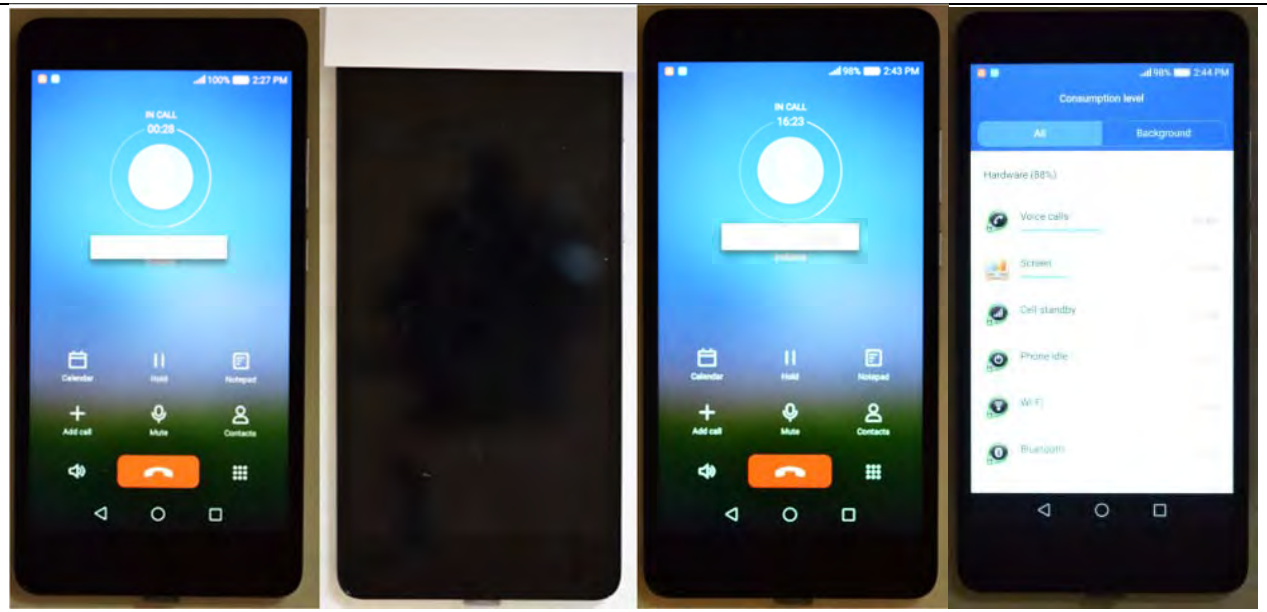
270	 When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 96%. The screen consumed 68.22% of the battery. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

271



When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 31.15% of the battery.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

272

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

273

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

274	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

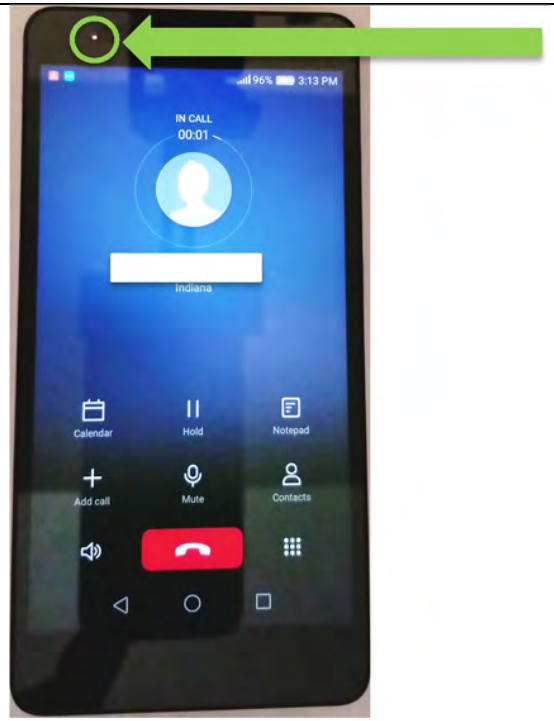
275	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

276

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Sensa is a range-detecting sensor.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

277

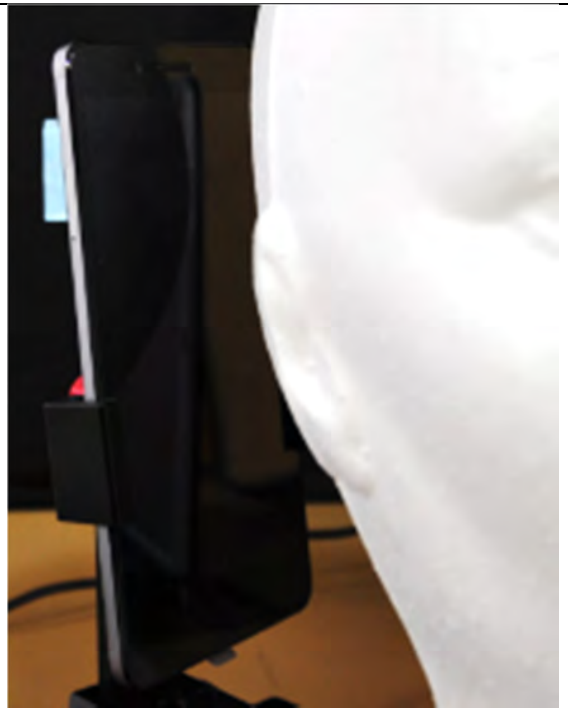


The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

The proximity sensor detects the range of a proximate object:

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.

278



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

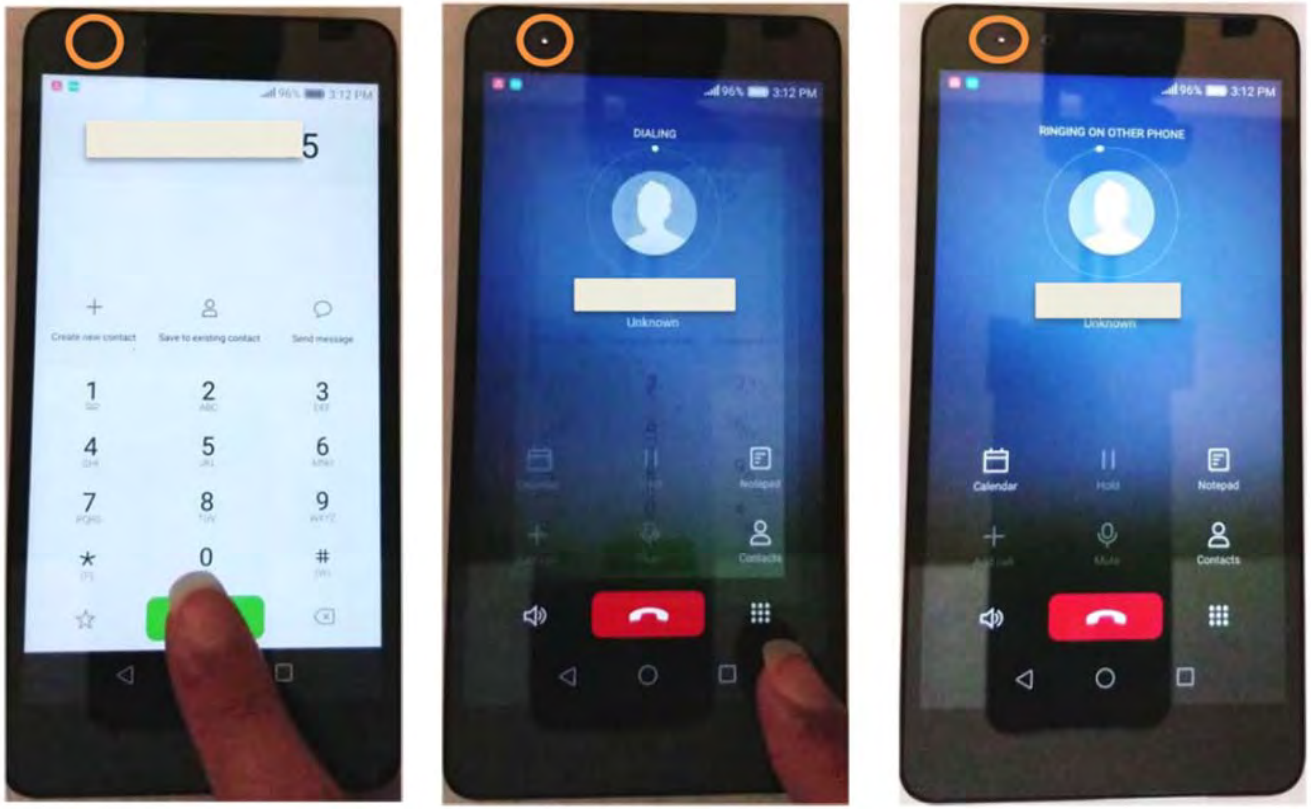
The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

279

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Sensa begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call. 

280

281

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

282	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

283

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call	The Huawei Sensa detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of

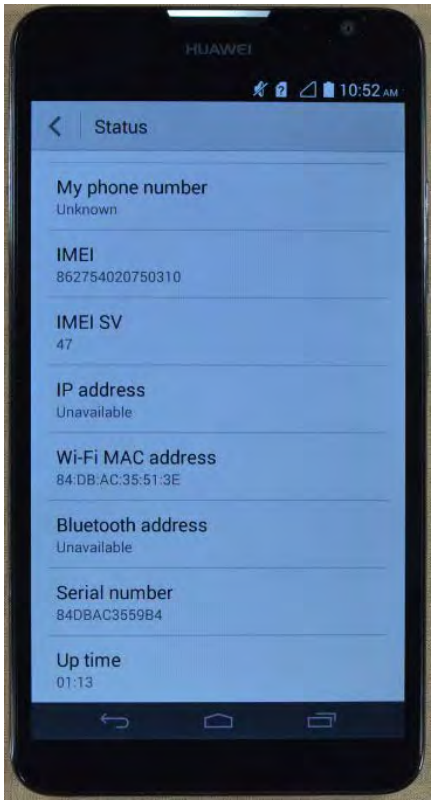
condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Sensa activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Sensa reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].

284

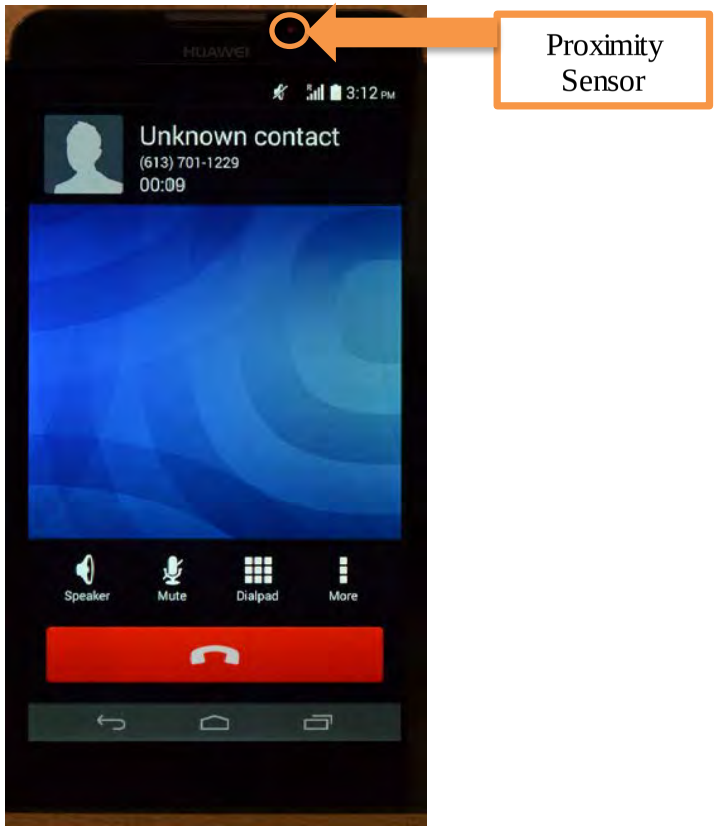
Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

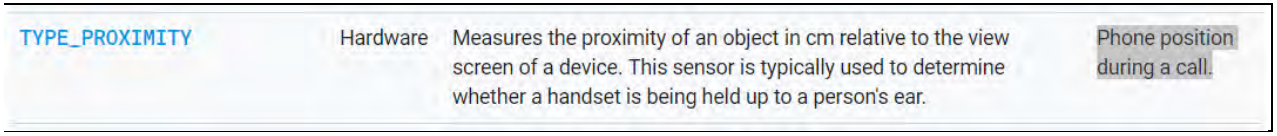
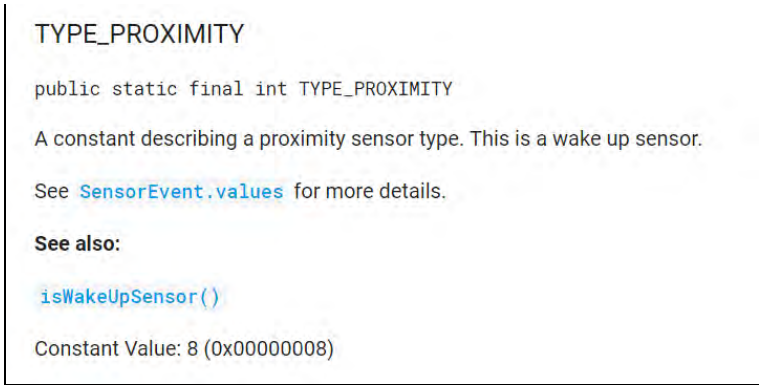
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].
and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Sensa is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

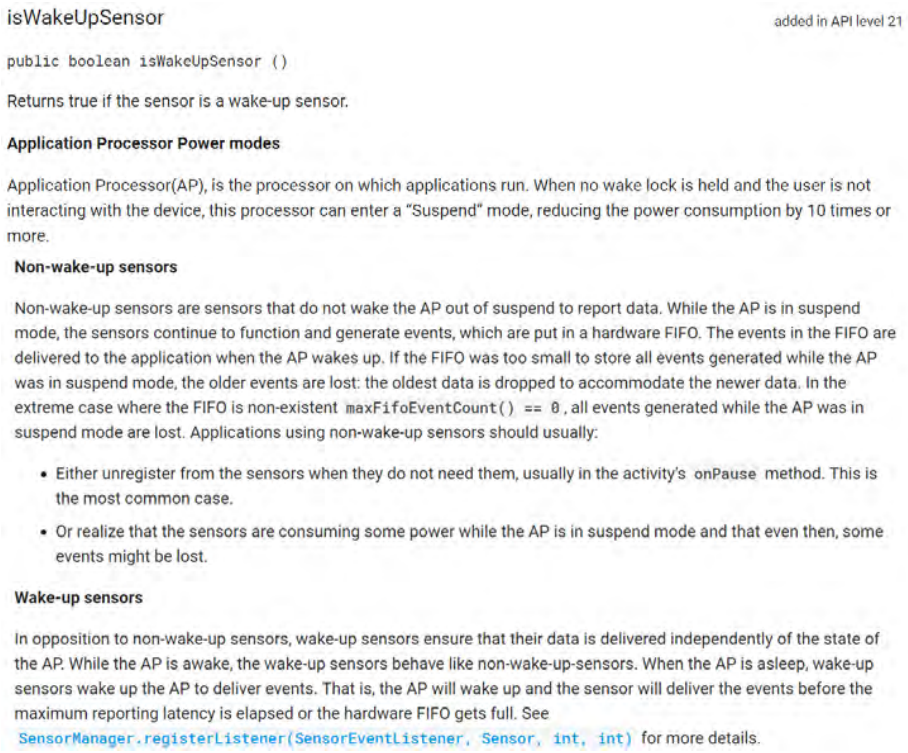
Asserted Claim Elements	Huawei Ascend Mate 2
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Ascend Mate 2 is a mobile station.  

[i] a display;	The Huawei Ascend Mate 2 includes a display.  *See http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00351749, last accessed Nov 29, 2018.*

[ii] a proximity sensor	The Huawei Ascend Mate 2 includes a proximity sensor. 

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Ascend Mate 2 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Ascend Mate 2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:  TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.  TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:



isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

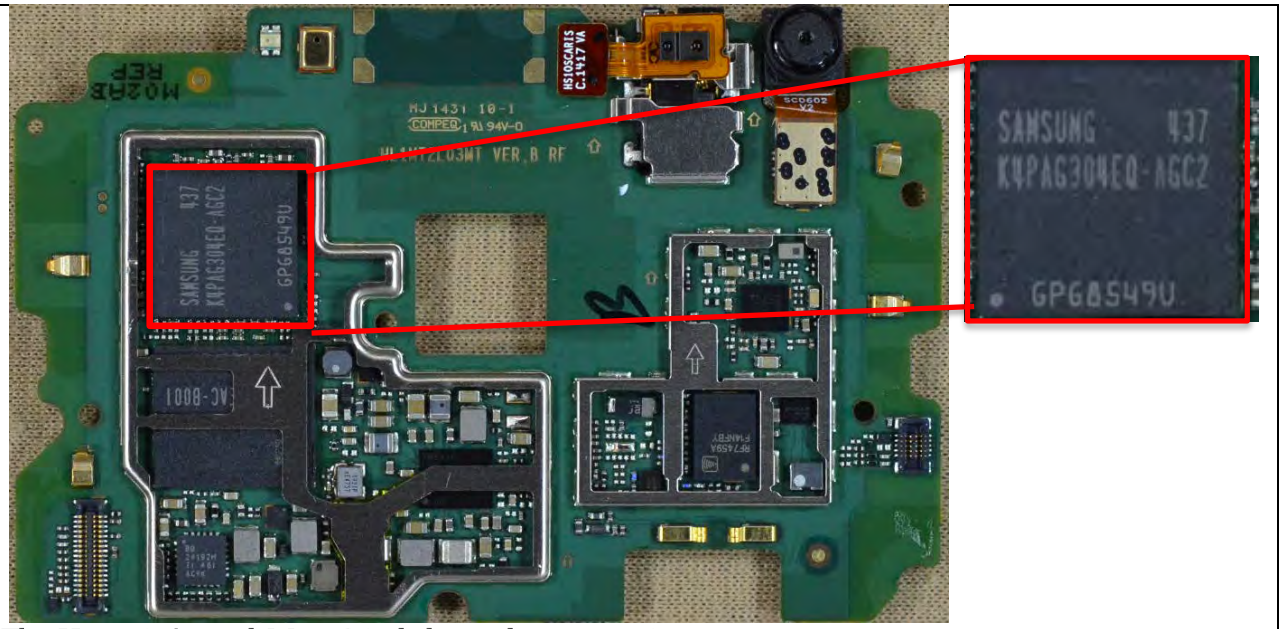
The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

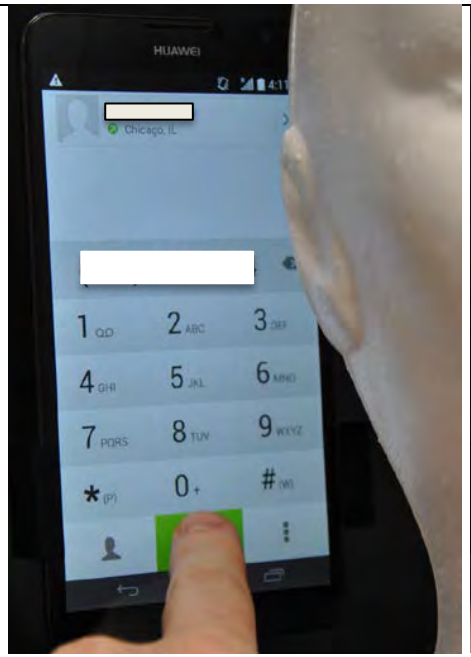
KOTLIN JAVA

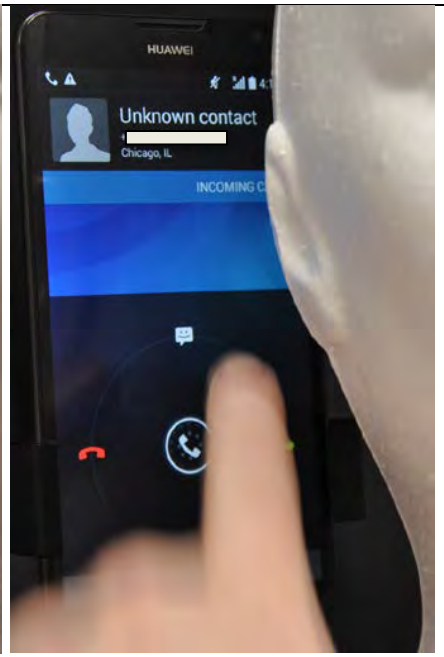
```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

[iv] a microprocessor adapted to:	 The Huawei Ascend Mate 2 includes at least one microprocessor. "1.6GHz Qualcomm Quad-Core MSM8928" *See https://www.cnet.com/products/huawei-ascend-mate2-4g/specs/, last accessed Nov. 29, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Ascend Mate 2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.





By way of example only, the Huawei Ascend Mate 2 is an Android phone. The Huawei Ascend Mate 2's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

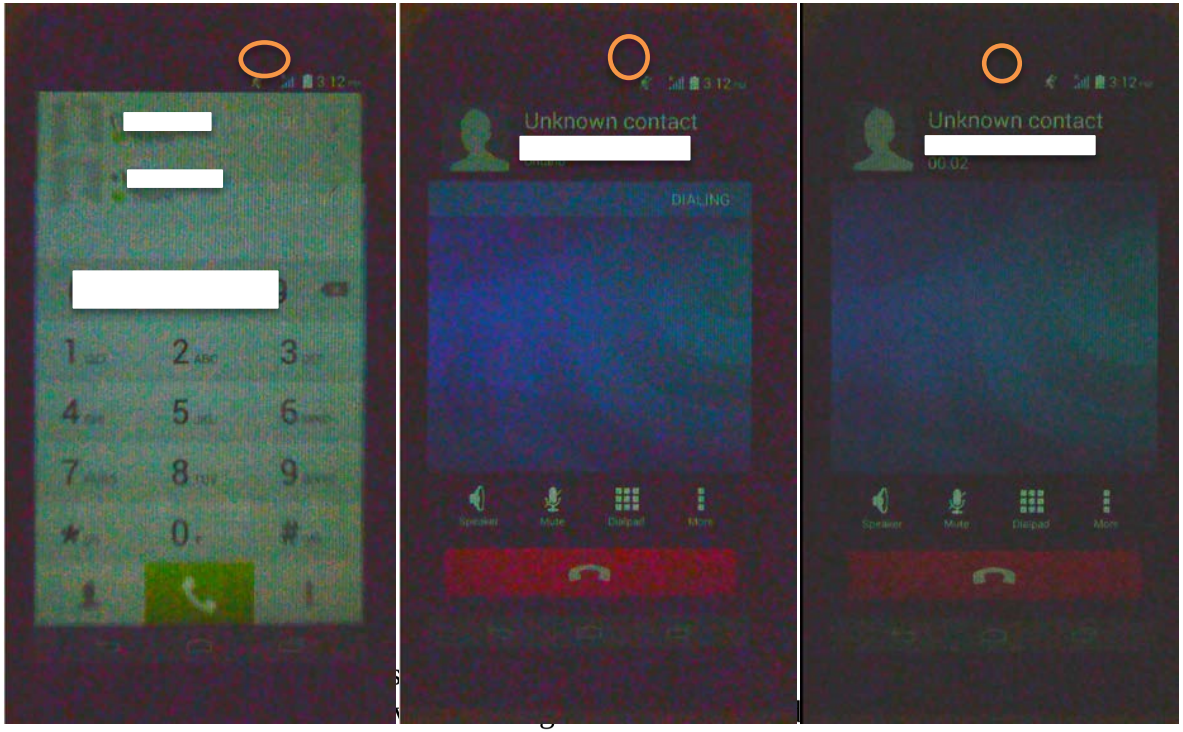
In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Ascend Mate 2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.

	The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.  By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

	See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018. In the Huawei Ascend Mate 2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Ascend Mate 2 is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

TYPE_PROXIMITY

`public static final int TYPE_PROXIMITY`

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

`public boolean isWakeUpSensor ()`

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

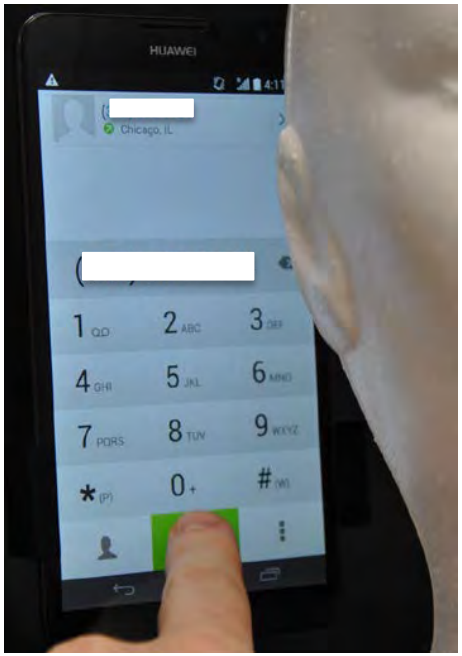
Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

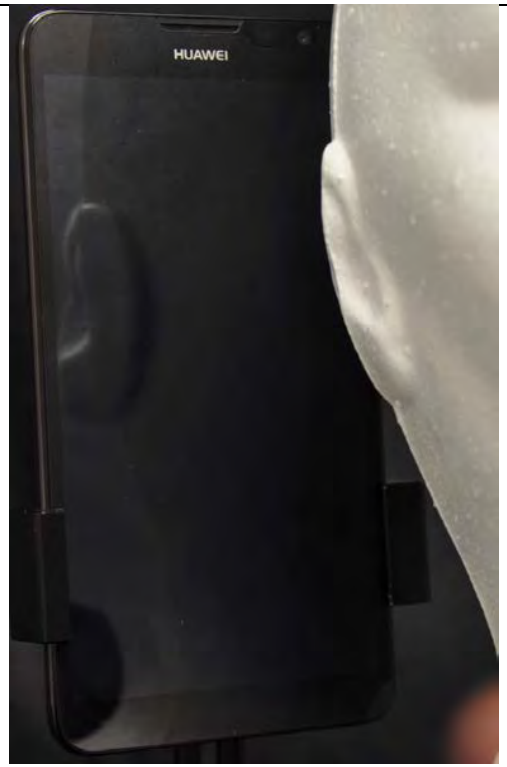
- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

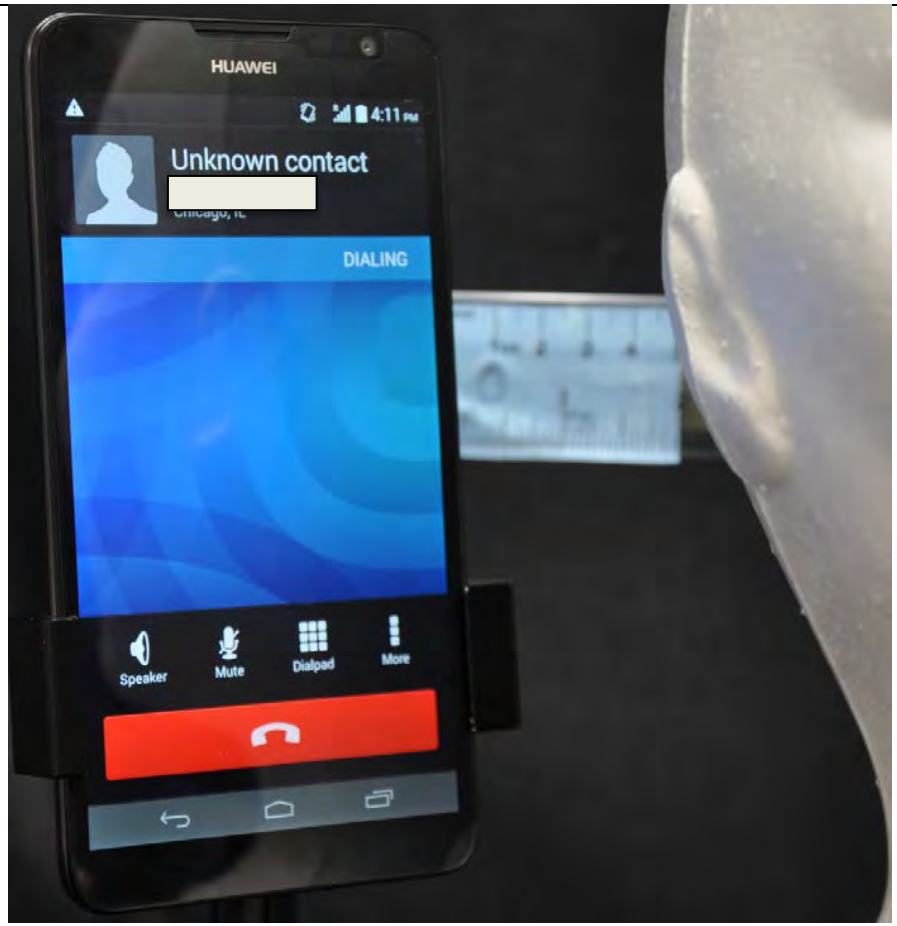
In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Ascend Mate 2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Ascend Mate 2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is darkened.



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 97%. The screen consumed 62% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 99%. The screen consumed 73% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

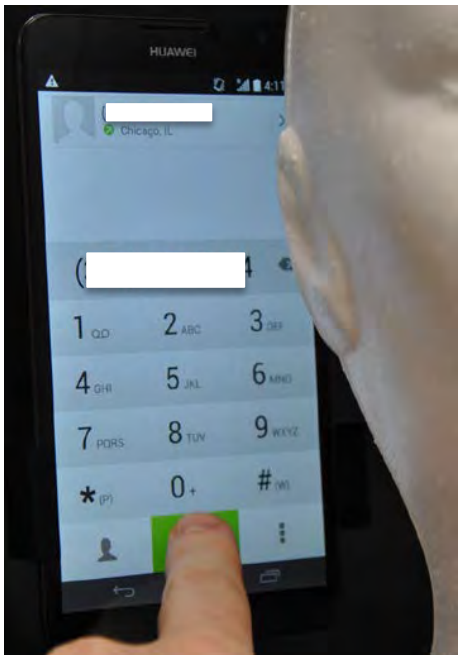
See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

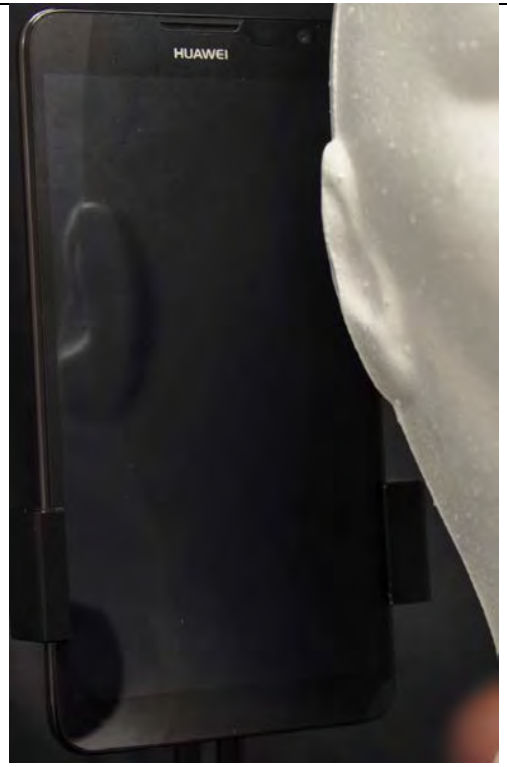
	In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)." Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

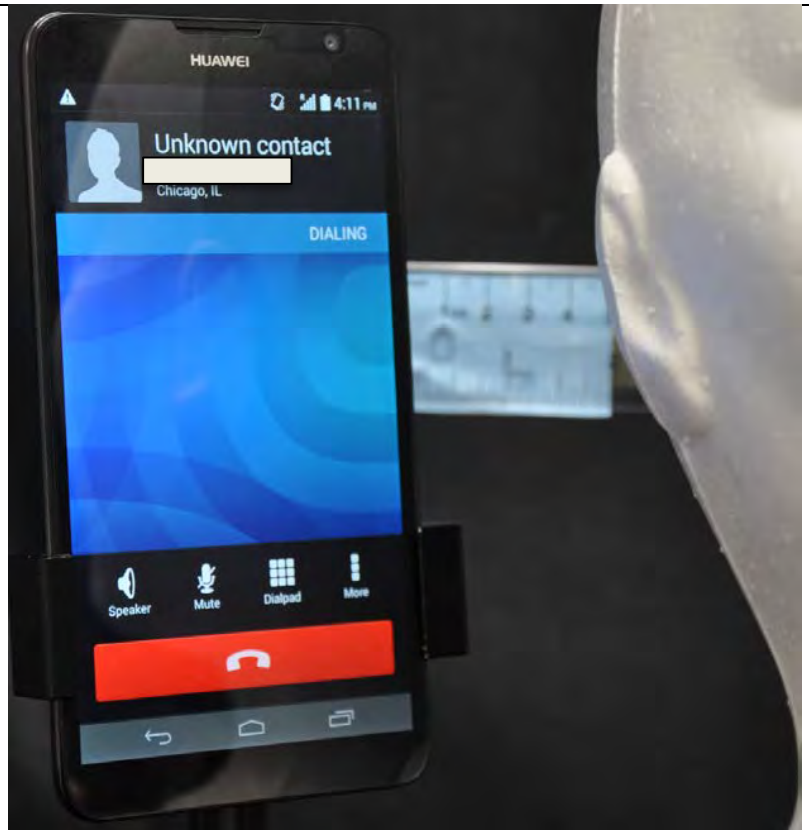
```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018*
2. The mobile station of claim 1,	*See* claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Ascend Mate 2 increases the power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Ascend Mate 2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is darkened.



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

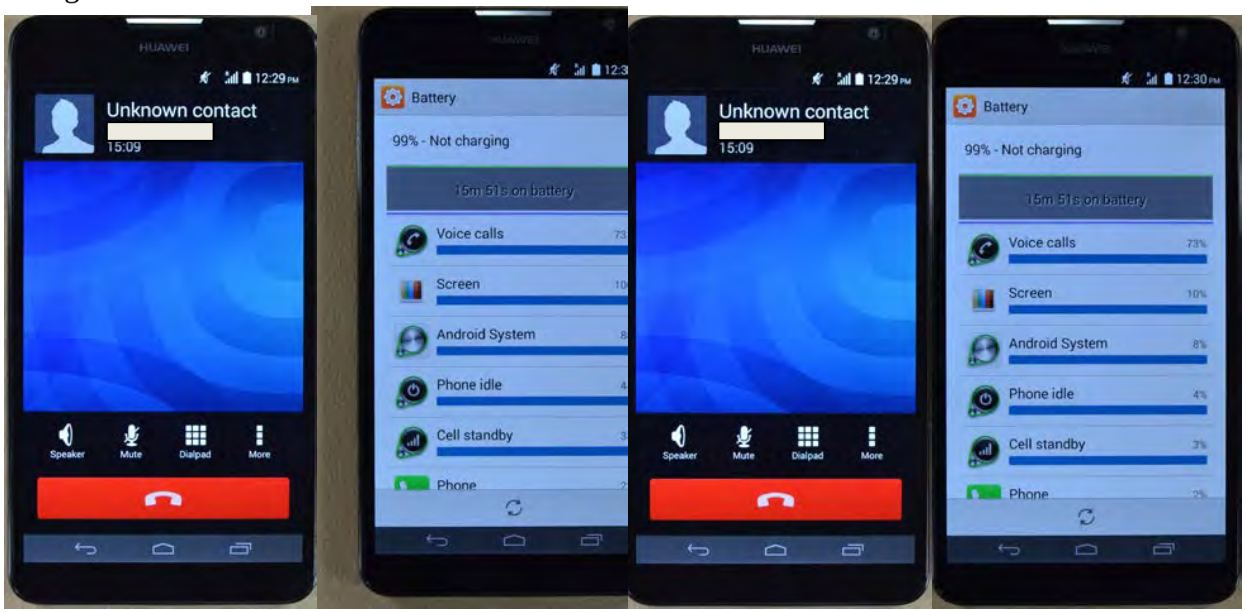
	 By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.

wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Ascend Mate 2 reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 97%. The screen consumed 62% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during call.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 99%. The screen consumed 73% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

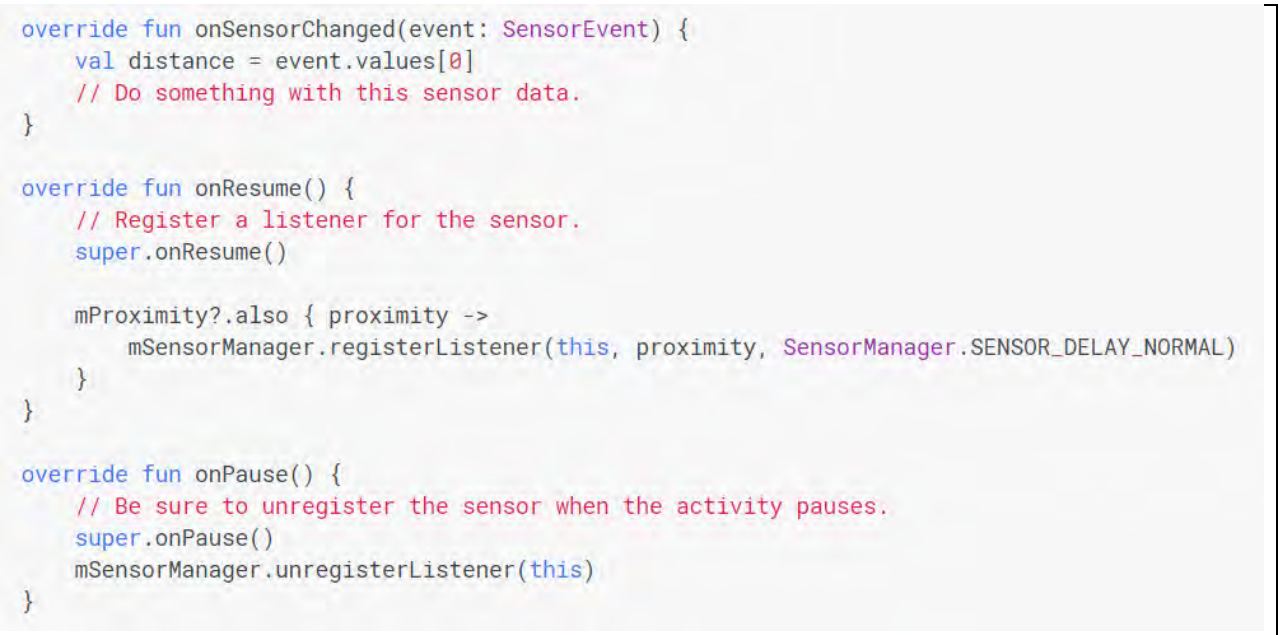
KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

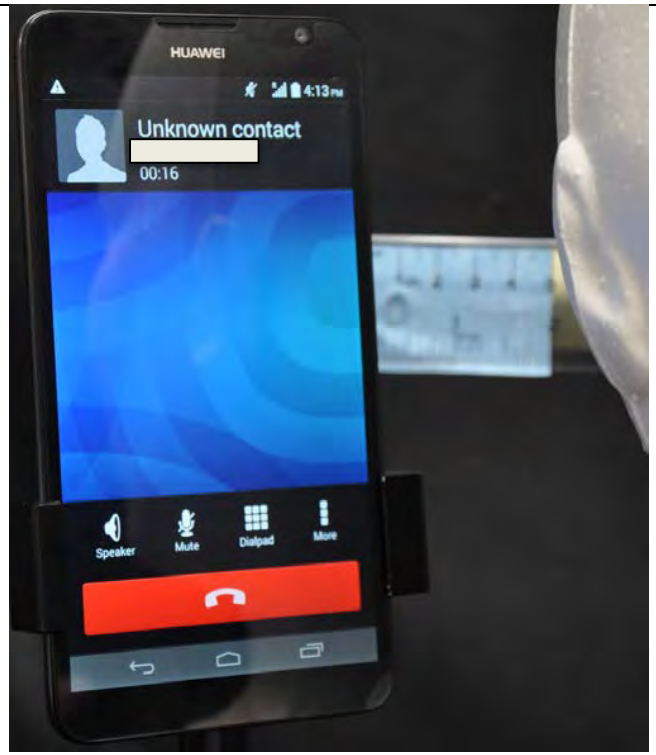
```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

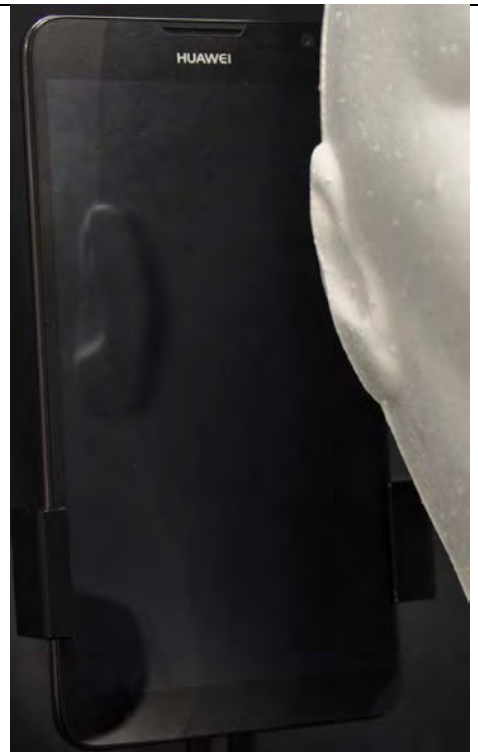
	 *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Ascend Mate 2 is a range-detecting sensor.

The proximity sensor detects the range of a proximate object:

The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 8 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

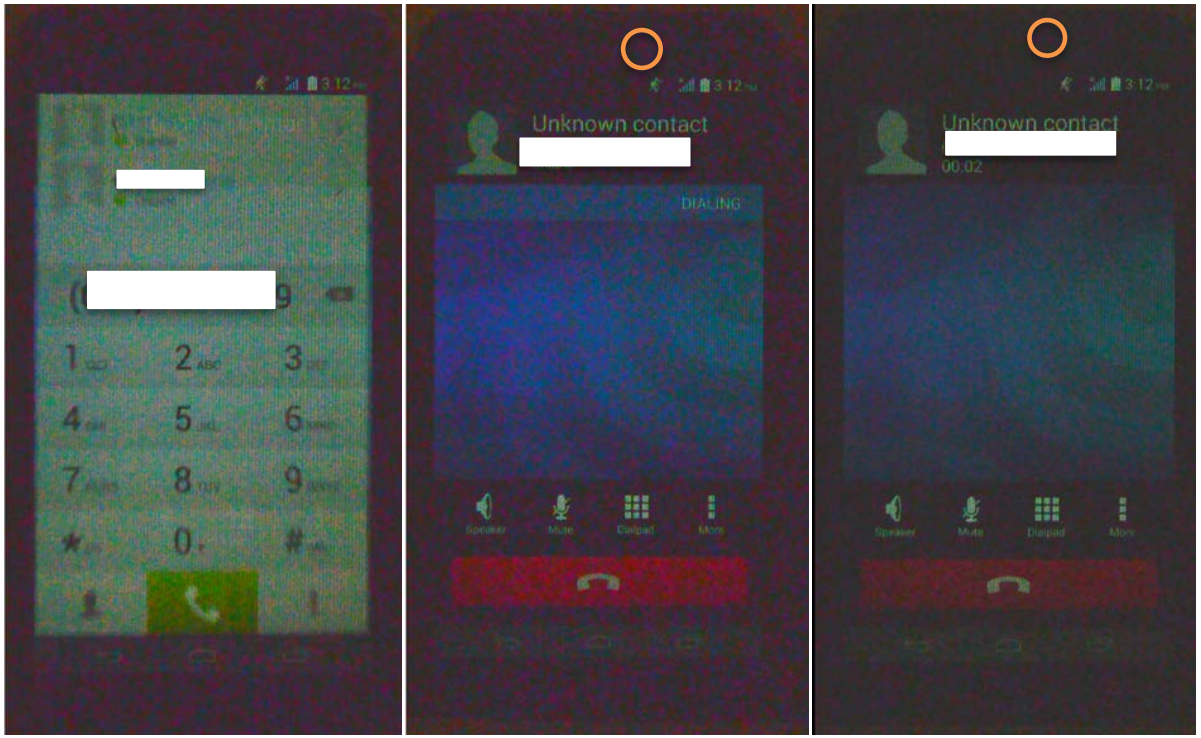
```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Ascend Mate 2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call. 

	In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different	The Huawei Ascend Mate 2 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.

from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	*See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Ascend Mate 2 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Ascend Mate 2 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].
and a microprocessor adapted to:	*See* claim[iv].

(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Ascend Mate 2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

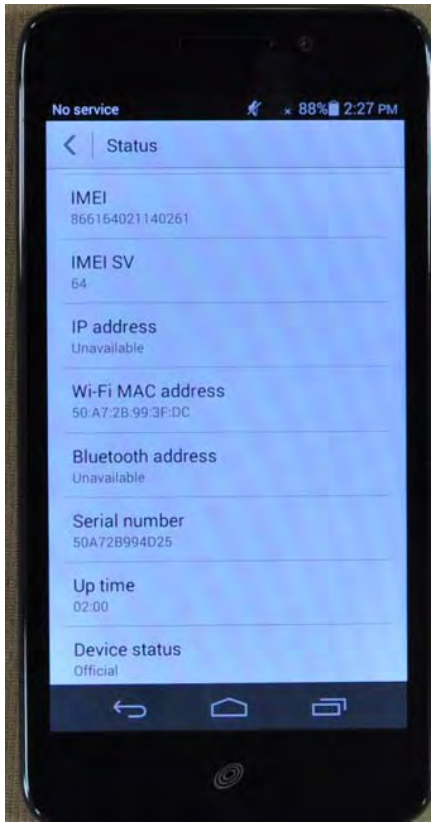
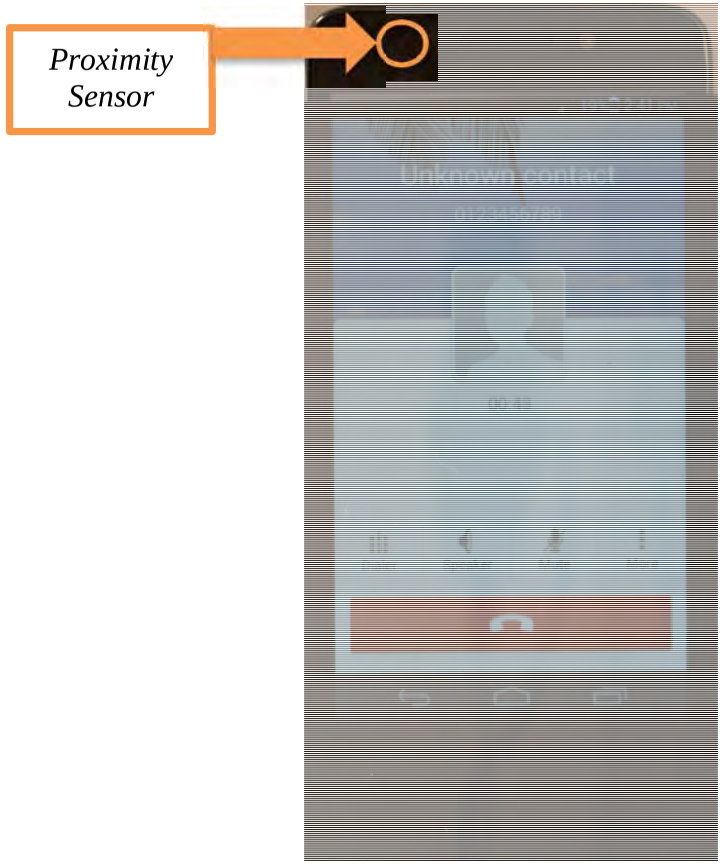
Asserted Claim Elements	Huawei Pronto
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Pronto is a mobile station.  

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Pronto includes a display.  *https://media.tracfone.com//wps/wcm/connect/d5a8bcde-3697-42f2-b08d-5590c1d90f6a/ST_QUAG_STH891G_ENG_F.pdf?MOD=AJPERES&CACHEID=d5a8bcde-3697-42f2-b08d-5590c1d90f6a p. 2, last accessed Nov. 28, 2018.*

334

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Pronto includes a proximity sensor. 
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being	The proximity sensor of the Huawei Pronto is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.

that an external object is proximate; and	By way of example only, the Huawei Pronto is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

336

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

337

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

338

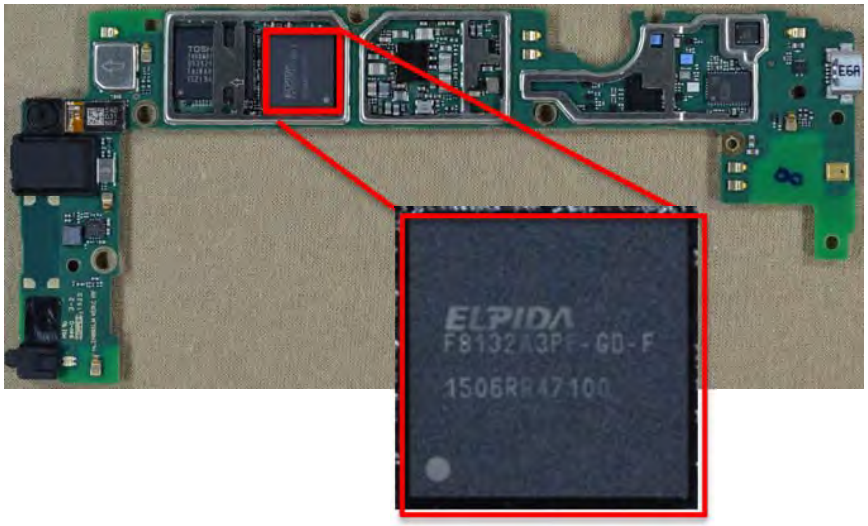
Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

339	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Pronto includes at least one microprocessor.

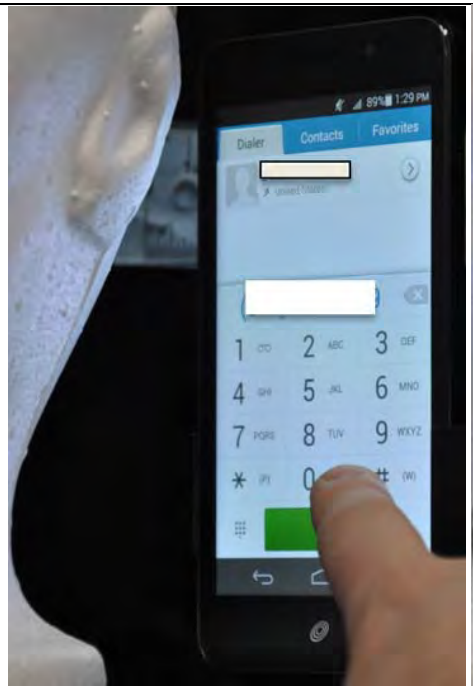
Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

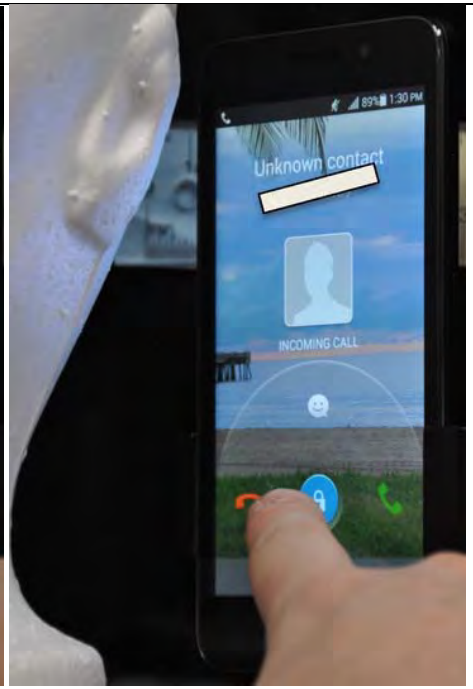
	  *Snapdragon 400 MSM8926*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Pronto is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.

341

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

342





By way of example only, the Huawei Pronto is an Android phone. The Huawei Pronto's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

343

```
/**
* Broadcast receiver to detect the outgoing calls.
*/
public class OutgoingReceiver extends BroadcastReceiver {
     public OutgoingReceiver() {
     }

     @Override
     public void onReceive(Context context, Intent intent) {
         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

         Toast.makeText(ctx,
           "Outgoing: "+number,
           Toast.LENGTH_LONG).show();
     }
}
```

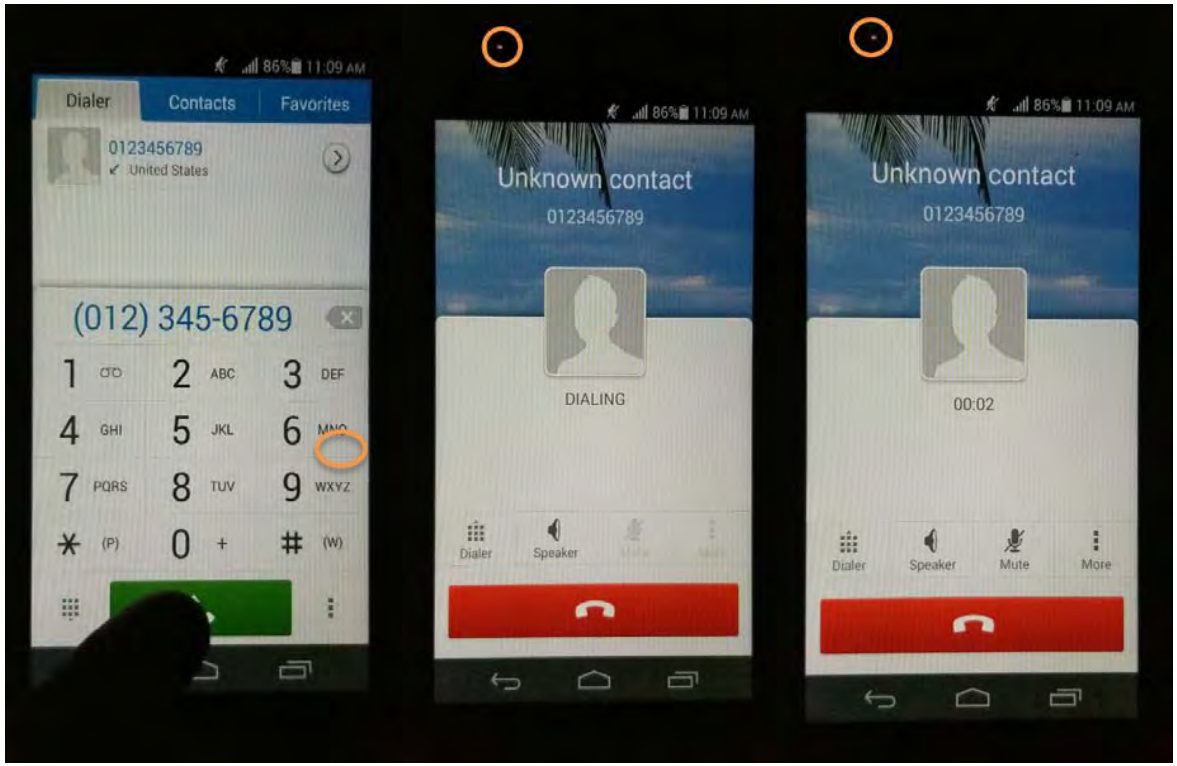
In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
* Listener to detect incoming calls.
*/
private class CallStateListener extends PhoneStateListener {
  @Override
  public void onCallStateChanged(int state, String incomingNumber) {
      switch (state) {
          case TelephonyManager.CALL_STATE_RINGING:
          // called when someone is ringing to this phone

          Toast.makeText(ctx,
                  "Incoming: "+incomingNumber,
                  Toast.LENGTH_LONG).show();
          break;
      }
  }
}
```

In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Pronto is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is

	activated. Third image shows the proximity sensor remains activated during the remainder of the call.  By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

345

346

TYPE_PROXIMITY

`public static final int TYPE_PROXIMITY`

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

`public boolean isWakeUpSensor ()`

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Pronto, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Pronto is adapted to receive the signal from the activated proximity sensor. When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.

347

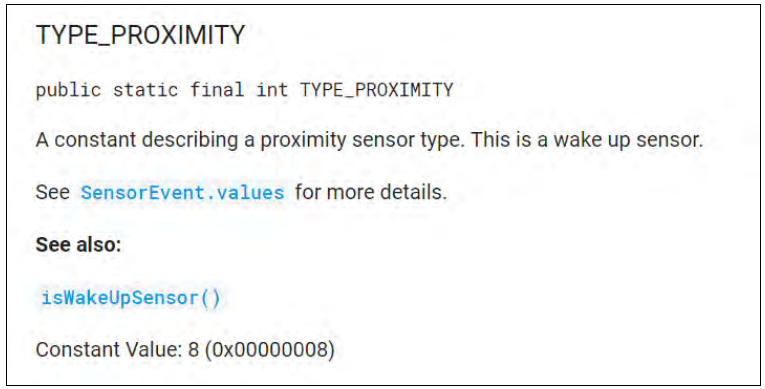
Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

348

349

By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:



TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

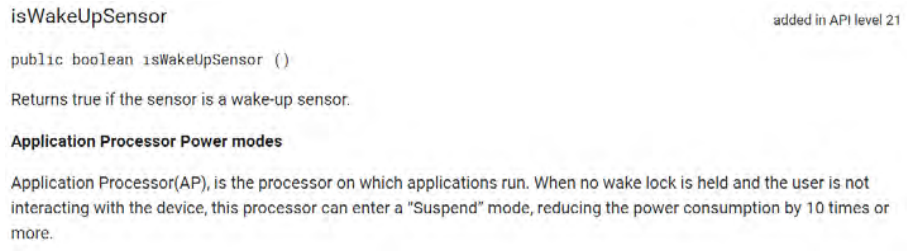
See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:



isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

350

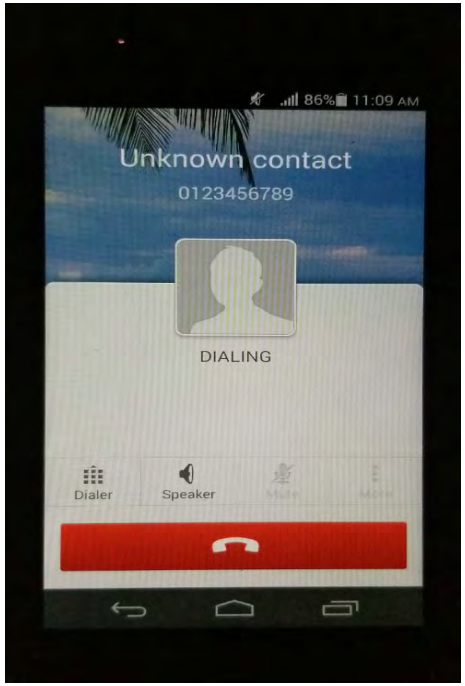
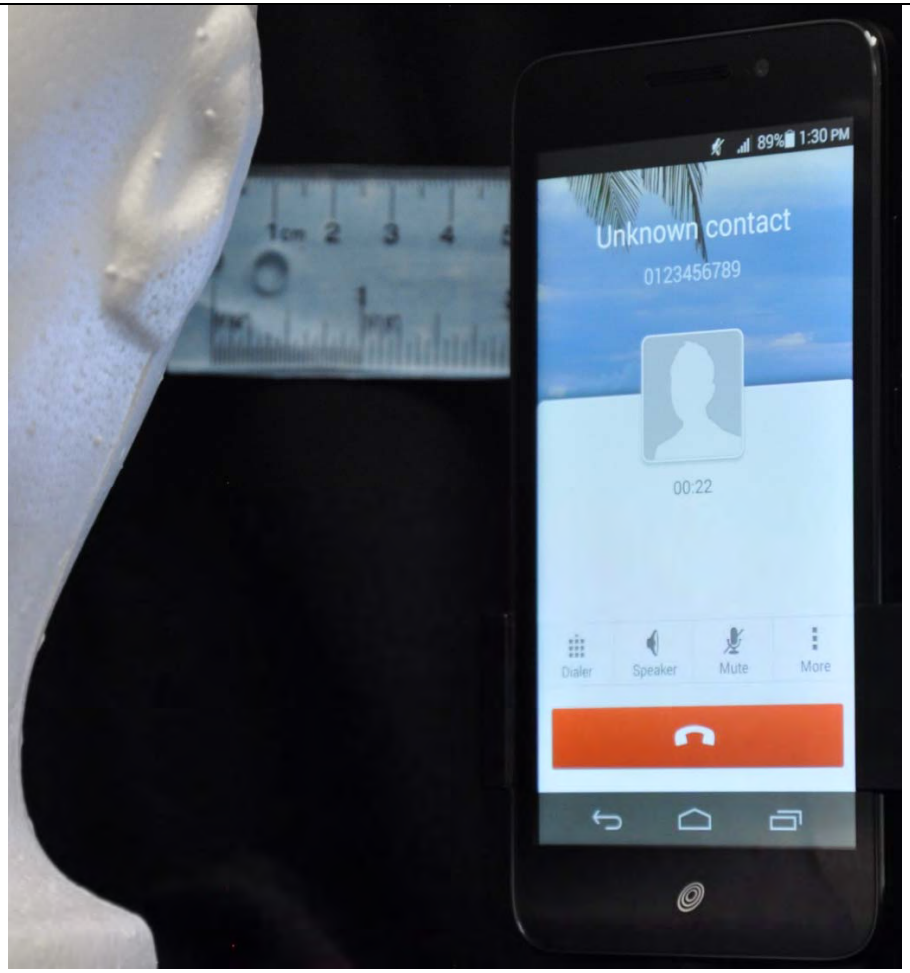
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Pronto is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the first image shows that the Huawei Pronto is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is darkened.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

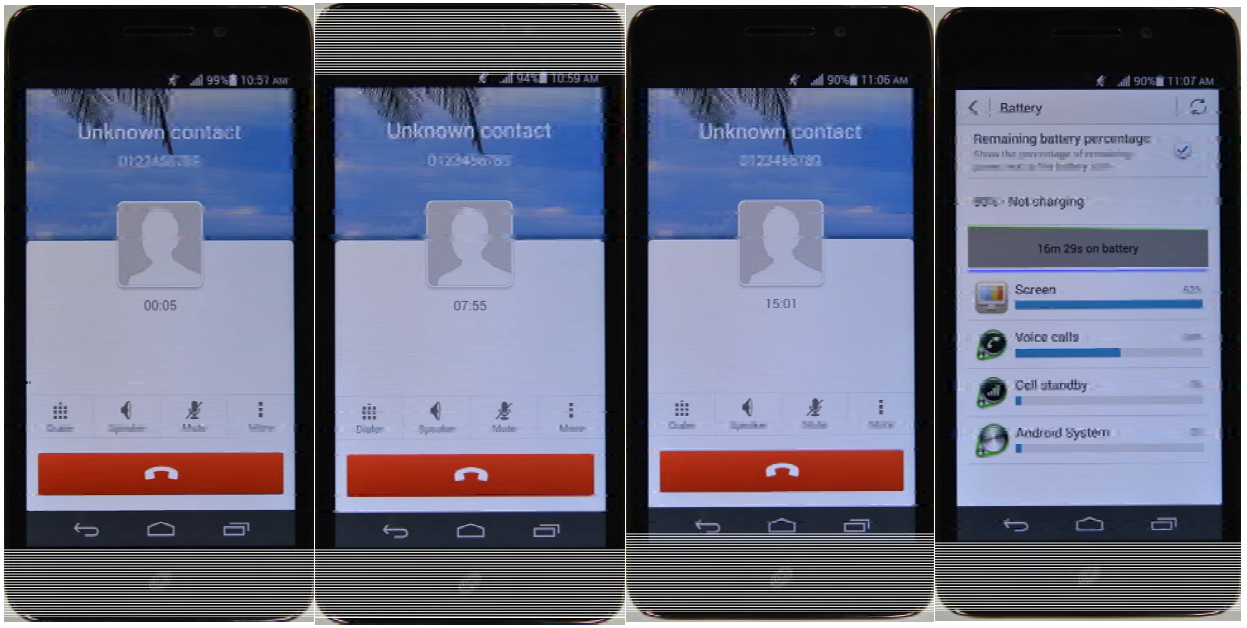
352

353



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.  When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 90%. The screen consumed 62% of the battery usage.

354

355	The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.  When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 95%. The screen consumed 79% of the battery usage. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

356

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

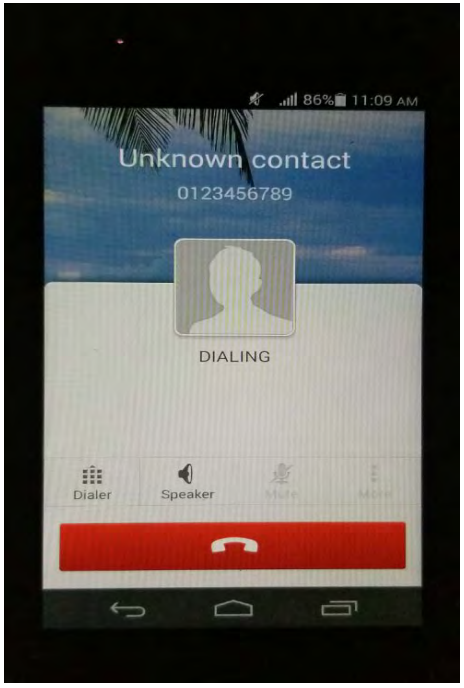
The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

357	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	358 ``` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Pronto increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active. By way of example only, the first image shows that the Huawei Pronto is proximate to an object, but no call has been initiated or answered, so the display is powered normally.  The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is darkened.

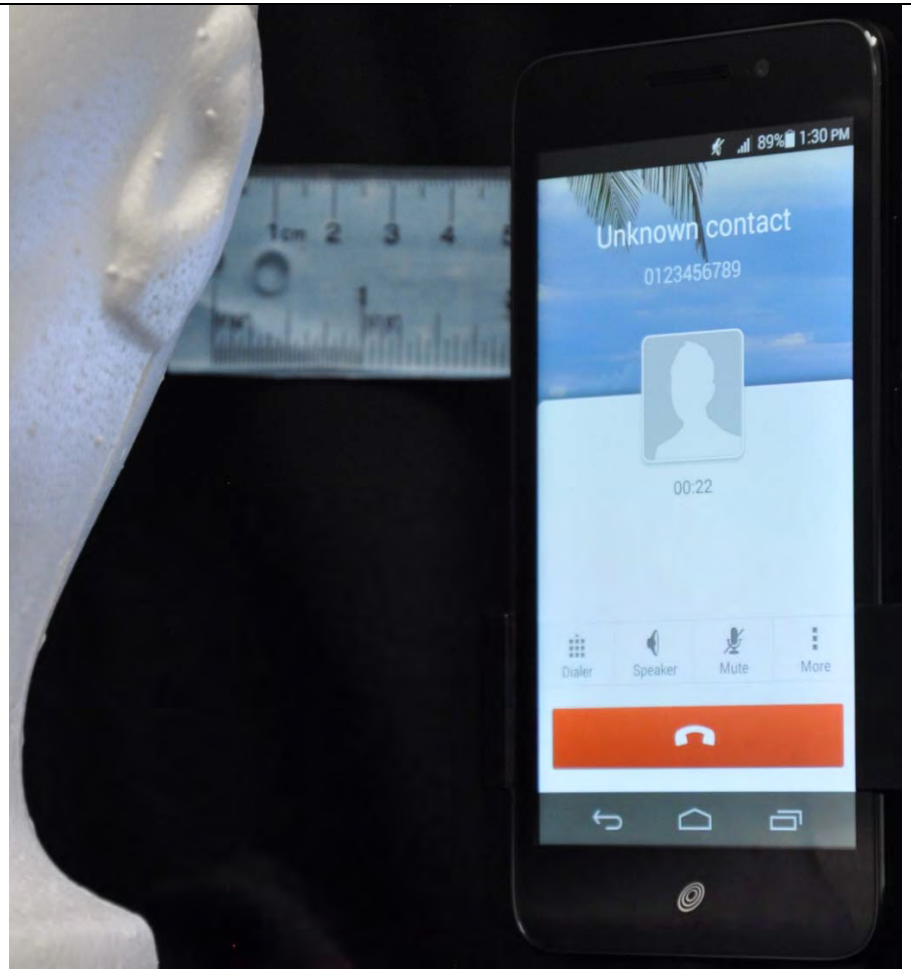
359

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	 The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

360

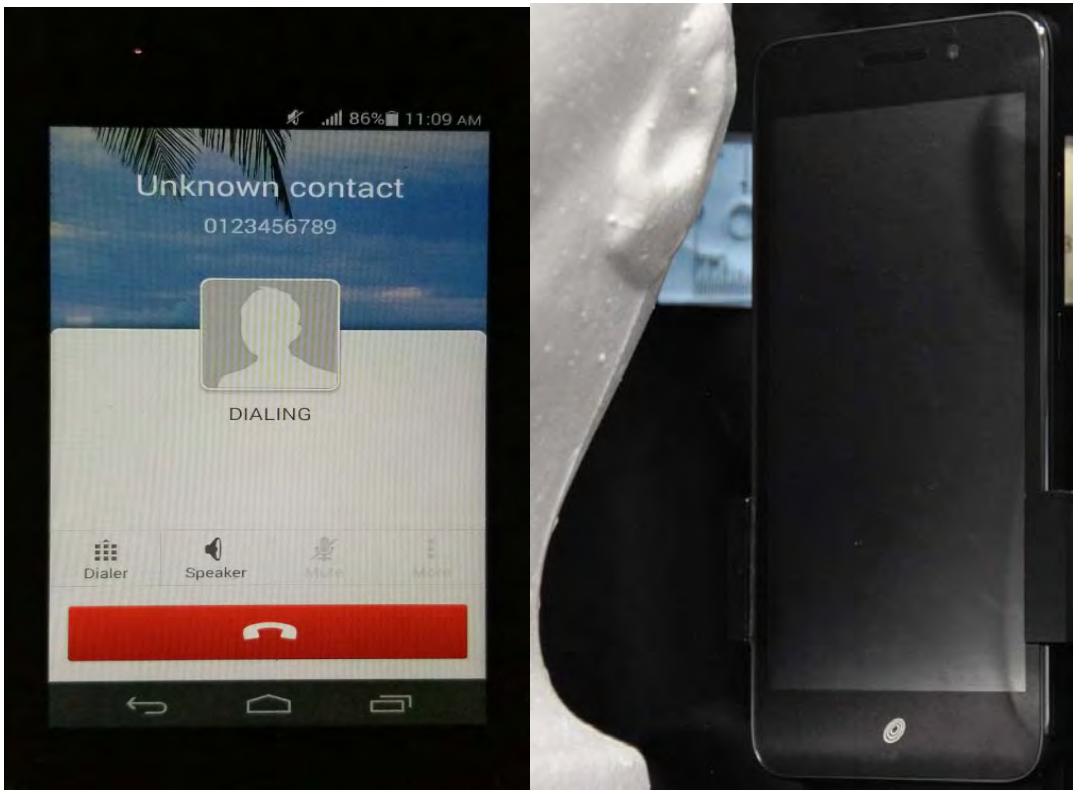
361



By way of further example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)."

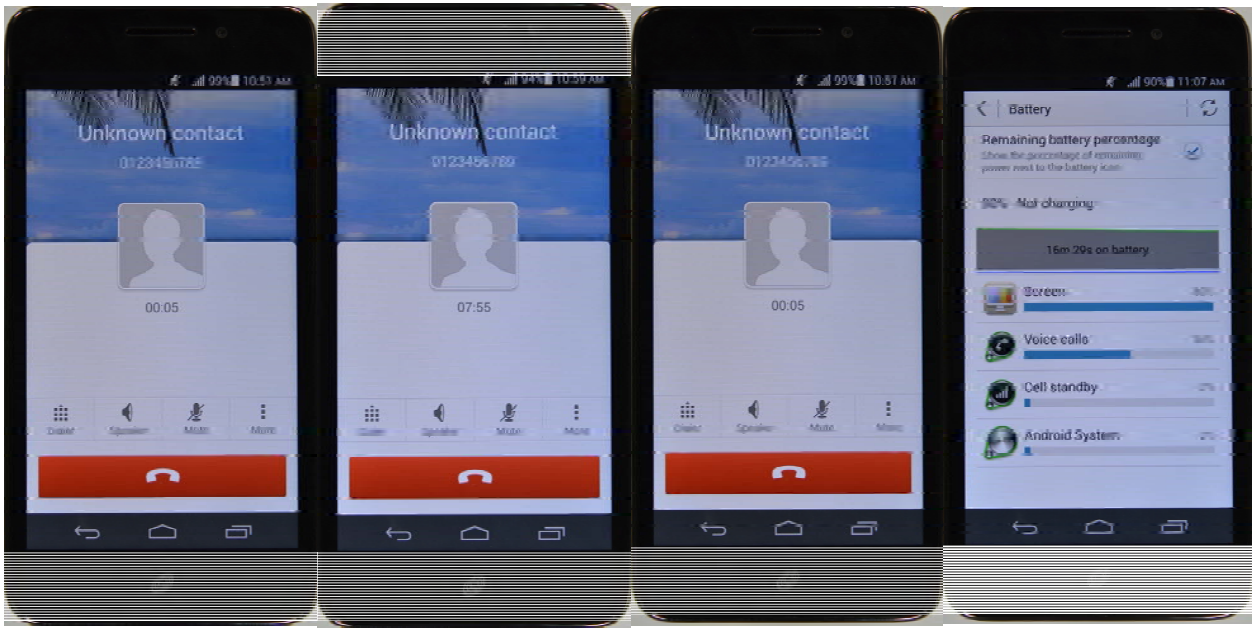
See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 30, 2018.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Pronto reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is turned off.  The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

362

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

	By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.  When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 90%. The screen consumed 62% of the battery usage. The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

363

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

364	 When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 95%. The screen consumed 79% of the battery usage. By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

365

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

366

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

367	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

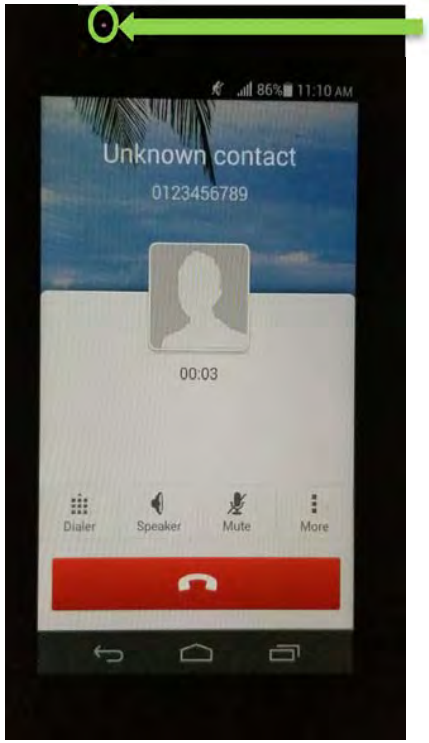
Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

368	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

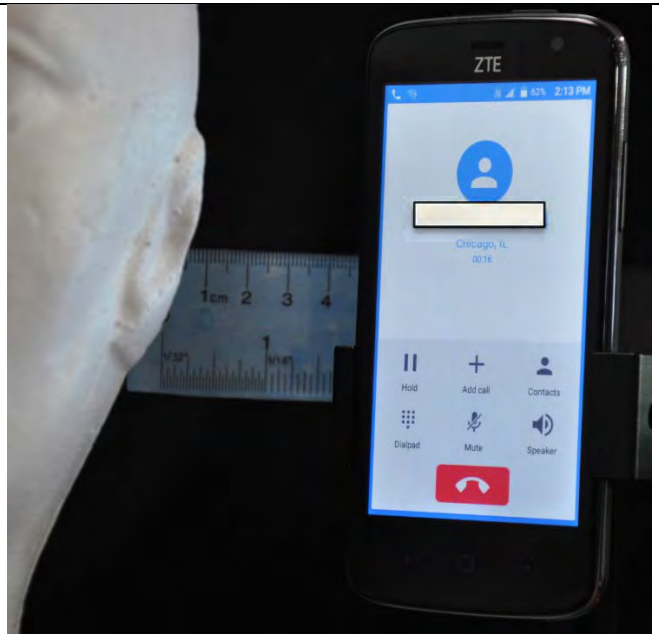
	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Pronto is a range-detecting sensor.

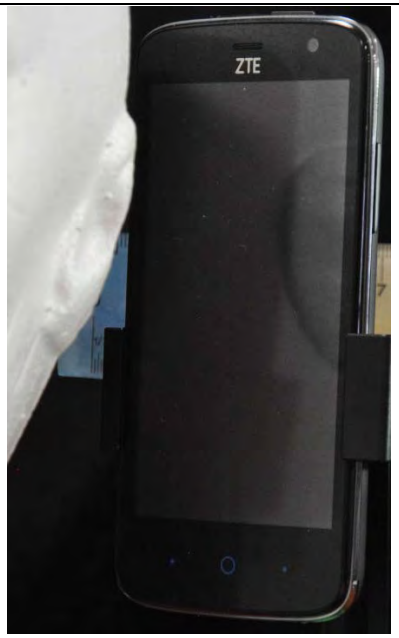
369

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

370	The proximity sensor detects the range of a proximate object:  *The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate* By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 6cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 3cm the display is off.

371





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

372

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

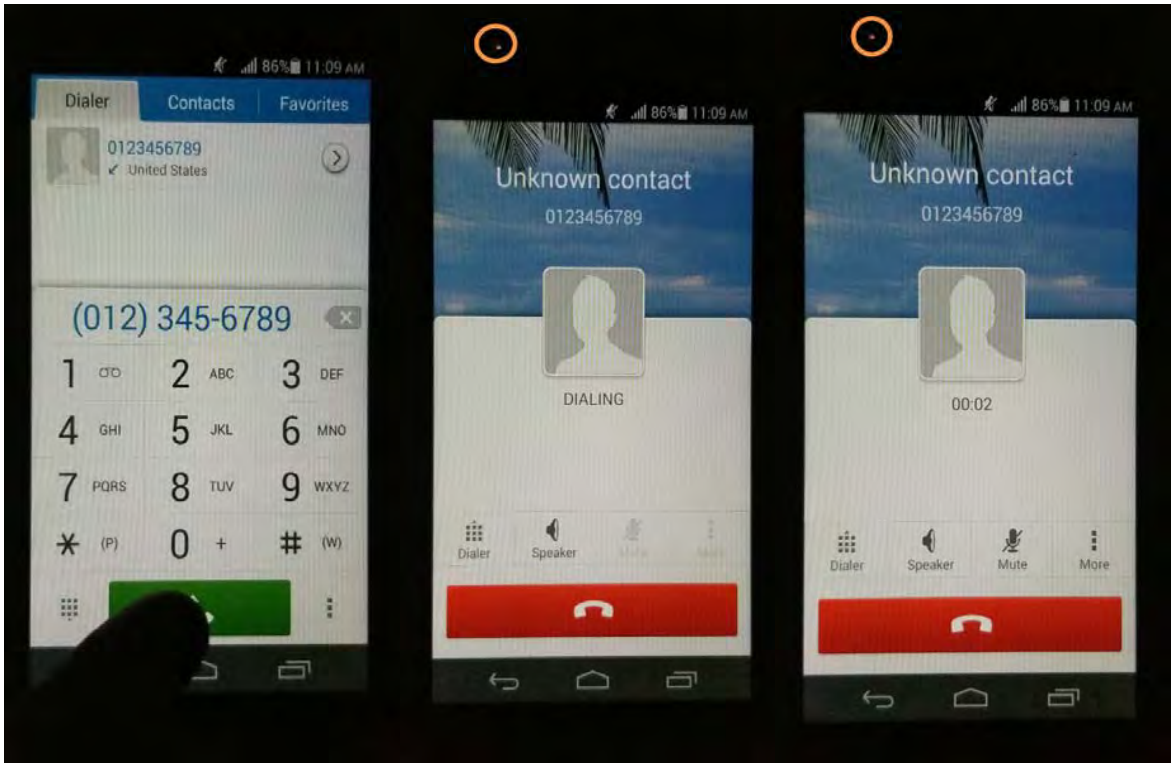
See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Pronto begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call. 

374

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

375	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

376

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different	The Huawei Pronto detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.

from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	*See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Pronto activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Pronto reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].
and a microprocessor adapted to:	*See* claim[iv].

Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554

(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Pronto is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Vision 2
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Vision 2 is a mobile station.  *See* https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%202%20User%20Guide.pdf , *last accessed December 6, 2018.*

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Vision 2 includes a display.  *See* https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%202%20User%20Guide.pdf , *last accessed December 6, 2018.*

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Vision 2 includes a proximity sensor.  *See* https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%202%20User%20Guide.pdf , *last accessed December 6, 2018.*
[iii] adapted to generate a signal indicative of the existence of a first	The proximity sensor of the Huawei Vision 2 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate

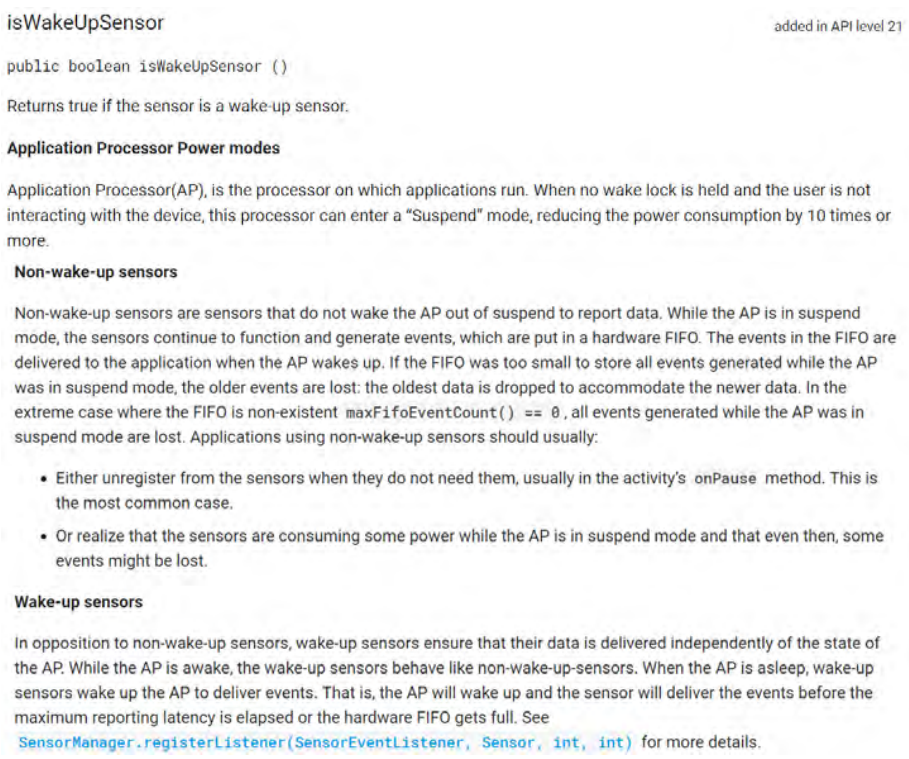
Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

condition, the first condition being that an external object is proximate; and	By way of example only, the Huawei Vision 2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY — Hardware — Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. — Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

382

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

383



isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to HUAWEI code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

384	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

385	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Vision 2 includes at least one microprocessor. |	| |---|---| | **Processor** | Qualcomm Snapdragon 200 | | **CPU Cores** | 2 | | **Processor Speed** | 1.2GHz | *See* https://www.phonerated.com/specs-huawei_vision_2, *last accessed December 6, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and	The microprocessor of the Huawei Vision 2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor: TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Vision 2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

388

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

389

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Vision 2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Vision 2 is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

390

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

392	**Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Vision 2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of further example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to HUAWEI code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

393

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

394

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

395

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Vision 2 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Vision 2, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Vision 2 reduces power to the display by turning off the display. By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

398

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
`TYPE_PROXIMITY`	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

400

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

401	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	402 ``` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Vision 2 is a range-detecting sensor. By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

403

	TYPE_PROXIMITY | Hardware | Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. | Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* And also, the code shows distance measurement: `override fun onSensorChanged(event: SensorEvent) {` `  val distance = event.values[0]` `  // Do something with this sensor data.` `}` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.* The Android Developer Code Website further explains: **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method. *See id.*

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Vision 2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei Code. Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

404

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

405	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Vision 2 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

406

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Vision 2 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Vision 2 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Vision 2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

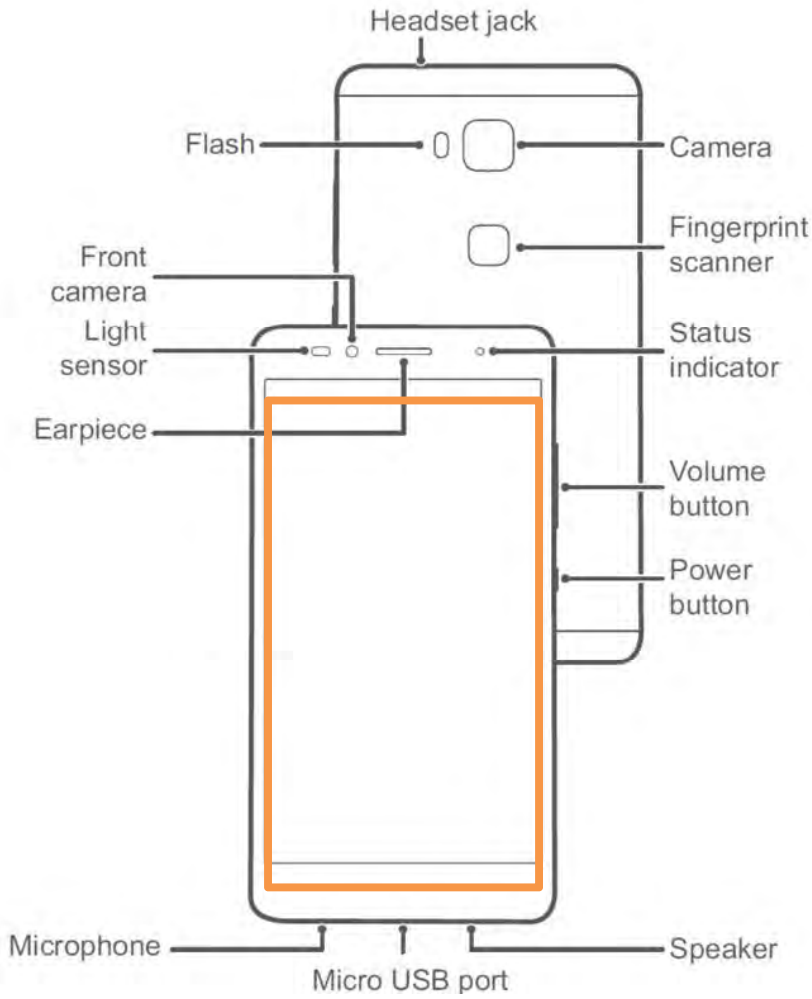
408

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Honor 5x
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Honor 5x is a mobile station.  *See* https://www.hihonor.com/us/product/10001646292075.html, *last accessed December 4, 2018.*

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Honor 5x includes a display, outlined in the figure below.



410

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	See http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00443446, p. 3, *last accessed December 4, 2018.*
[ii] a proximity sensor	The Huawei Honor 5x includes a proximity sensor.  *See* https://www.youtube.com/watch?v=aoCotiH4yrQ, *last accessed December 4, 2018.*

411

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Honor 5x is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Honor 5X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.*

412

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

414

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

415	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Honor 5x includes at least one microprocessor. Combination of 2 GB RAM and a Qualcomm 64-bit Octa-core processor for a smooth responsive experience *See* https://www.hihonor.com/us/product/10001646292075.html, *last accessed December 4, 2018.*
(a) determine, without using the proximity sensor, the existence of a	The microprocessor of the Huawei Honor 5x is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Huawei Honor 5X is an Android phone. The Huawei Honor 5X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `  public OutgoingReceiver() {` `  }` `  @Override` `  public void onReceive(Context context, Intent intent) {` `  String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `  Toast.makeText(ctx,` `  "Outgoing: "+number,` `  Toast.LENGTH_LONG).show();` `  }` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

417

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Honor 5x is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

418

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

419

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor 5x, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Honor 5X is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

420

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

421

TYPE_PROXIMITY

`public static final int TYPE_PROXIMITY`

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

`public boolean isWakeUpSensor ()`

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

422

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In the Huawei Honor 5X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Honor 5X is adapted to reduce power to the display if (i) the microprocessor determines that a telephone call is active and (ii) the signal indicates the proximity of the external object. By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

423

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

424	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

425	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Honor 5x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Honor 5x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Honor 5x reduces power to the display by turning off the display. By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

427

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

428

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

429

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

430

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

431	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Honor 5X is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

433	

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Honor 5X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

434

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

435	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	436 ``` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Honor 5x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Honor 5x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Honor 5x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

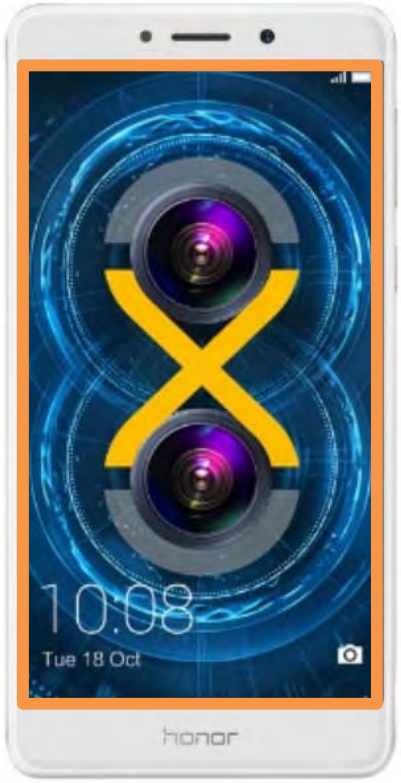
and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 5x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

438

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Honor 6x
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Honor 6x is a mobile station.  *See* https://www.hihonor.com/us/product/10001911052020.html#1000000100082, *last accessed December 4, 2018.*

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Honor 6x includes a display, outlined in the figure below.  *See* https://www.hihonor.com/us/product/10001911052020.html#1000000100082, *last accessed December 4, 2018.*

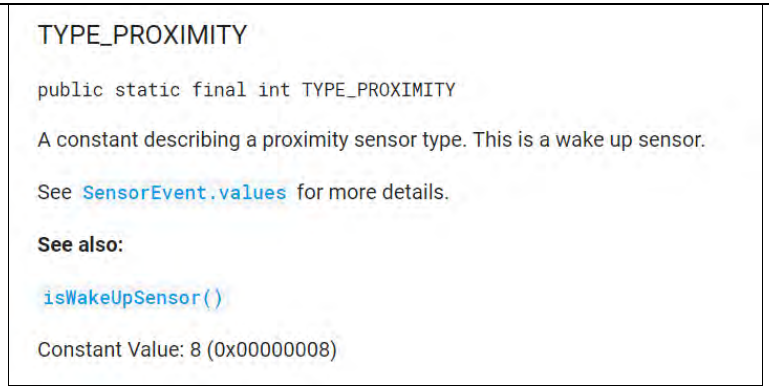
440

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Honor 6x includes a proximity sensor. **SPECIFICATIONS** SENSOR Hall effect sensor,Fingerprint sensor, Proximity sensor, Ambient light sensor, Compass, Accelerometer, Phone status indicator *See* https://www.hihonor.com/us/product/10001911052020.html#1000000100082, *last accessed December 4, 2018.*
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Honor 6X is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Honor 6X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

441

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554



TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

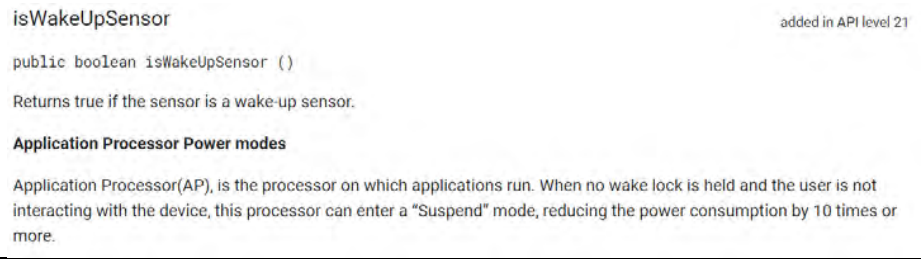
See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:



isWakeUpSensor

added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

442

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

443

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

444

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

445	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Honor 6x includes at least one microprocessor. CPU Model:Huawei Kirin 655,Octa-Core (4x2.1GHz+4x1.7GHz) *See* https://www.hihonor.com/us/product/10001646292075.html, *last accessed December 4, 2018.*

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 6X is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Huawei Honor 6X is an Android phone. The Huawei Honor 6X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `  public OutgoingReceiver() {` `  }` `  @Override` `  public void onReceive(Context context, Intent intent) {` `  String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `  Toast.makeText(ctx,` `  "Outgoing: "+number,` `  Toast.LENGTH_LONG).show();` `  }` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

447

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Honor 6x is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor 6x, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Honor 6X is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

450

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

451

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

452

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In the Huawei Honor 6X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Honor 6X is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

453

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

454	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

455	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	456 `override fun onSensorChanged(event: SensorEvent) {` `  val distance = event.values[0]` `  // Do something with this sensor data.` `}` `override fun onResume() {` `  // Register a listener for the sensor.` `  super.onResume()` `  mProximity?.also { proximity ->` `  mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)` `  }` `}` `override fun onPause() {` `  // Be sure to unregister the sensor when the activity pauses.` `  super.onPause()` `  mSensorManager.unregisterListener(this)` `}` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Honor 6x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Honor 6x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Honor 6X reduces power to the display by turning off the display. By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate: TYPE_PROXIMITY `public static final int TYPE_PROXIMITY` A constant describing a proximity sensor type. This is a wake up sensor. See `SensorEvent.values` for more details. **See also:** `isWakeUpSensor()` Constant Value: 8 (0x00000008)

457

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

458

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

459

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

460	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Honor 6X is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors: TYPE_PROXIMITY | Hardware | Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. | Phone position during a call.

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

462	*See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* And also, the code shows distance measurement: `override fun onSensorChanged(event: SensorEvent) {` `  val distance = event.values[0]` `  // Do something with this sensor data.` `}` *See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.* The Android Developer Code Website further explains: **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method. *See id.*

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Honor 6X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

463

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

464	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Honor 6x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

465

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Honor 6x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Honor 6x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 6x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Honor 7x
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Honor 7X is a mobile station.  *See* https://www.hihonor.com/us/product/10001933050073.html#1000000100062, *last accessed December 5, 2018.*

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Honor 7X includes a display, outlined in the figure below.  *See* https://www.hihonor.com/us/product/10001933050073.html#1000000100062, *last accessed December 5, 2018.*

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Honor 7x includes a proximity sensor. **SPECIFICATIONS** SENSOR Fingerprint sensor, Proximity sensor, Ambient light sensor, Compass, Gravity sensor, Phone status indicator *See* https://www.hihonor.com/us/product/10001933050073.html#1000000100062, *last accessed December 5, 2018.*
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Honor 7x is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate. By way of example only, the Huawei Honor 7X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY | Hardware | Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. | Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

470

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

TYPE_PROXIMITY

`public static final int TYPE_PROXIMITY`

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor — added in API level 21

`public boolean isWakeUpSensor ()`

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

472

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

473	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

474

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Honor 7x includes at least one microprocessor. CPU Model:Kirin 695,CPU Cores:Octa-Core(4*2.36GHz+4*1.7GHz) *See* https://www.hihonor.com/us/product/10001933050073.html#1000000100062, *last accessed December 5, 2018.*
(a) determine, without using the proximity sensor, the existence of a	The microprocessor of the Huawei Honor 7X is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Huawei Honor 7X is an Android phone. The Huawei Honor 7X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `  public OutgoingReceiver() {` `  }` `  @Override` `  public void onReceive(Context context, Intent intent) {` `  String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `  Toast.makeText(ctx,` `  "Outgoing: "+number,` `  Toast.LENGTH_LONG).show();` `  }` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

476

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Honor 7X is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

478

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor 7X, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Honor 7X is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

480

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

481

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In the Huawei Honor 7X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Honor 7X is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

482

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

483

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

484	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

485

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Honor 7x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Honor 7x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Honor 7X reduces power to the display by turning off the display. By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

486

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

TYPE_PROXIMITY

public static final int TYPE_PROXIMITY

A constant describing a proximity sensor type. This is a wake up sensor.

See SensorEvent.values for more details.

See also:

isWakeUpSensor()

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the is WakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

488

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

489	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

490	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Honor 7X is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:

491

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

492

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Honor 7X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

493

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

494	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Honor 7x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Honor 7x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Honor 7x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 7x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

Asserted Claim Elements	Huawei Honor 8
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Honor 8 is a mobile station.  *See* https://www.hihonor.com/global/products/smartphone/honor8/, *last accessed December 5, 2018.*

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Honor 8 includes a display, outlined in the figure below.  *See* https://www.hihonor.com/global/products/smartphone/honor8/, *last accessed December 5, 2018.*

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Honor 8 includes a proximity sensor. *See, e.g.*, the following discussion on removing the proximity sensor from an Honor 8: Repair guide : Proximity and Light Sensor Huawei Honor 8  (https://medias.sosav.fr/images/guides/big/8dgtg0ocfl6nc0w.jpg?version=1.8) Step 28 Use the pair of tweezers and grab the proximity/light sensor and remove it.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	 (https://medias.sosav.fr/images/guides/big/njd3bn2rf7v4nah.jpg?version=1.8) Step 29 The proximity/light sensor of your Honor 8 is now disassembled. You can replace it with a new one if needed. *See* https://www.sosav.com/guides/mobiles/huawei/honor-8/proximity-light-sensor/, *last accessed December 5, 2018.*
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an	The proximity sensor of the Huawei Honor 8 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate By way of example only, the Huawei Honor 8 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their

501

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

external object is proximate; and	hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008) *See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

502

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

503

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:	

504

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

505

KOTLIN JAVA

```kotlin
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

506

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*

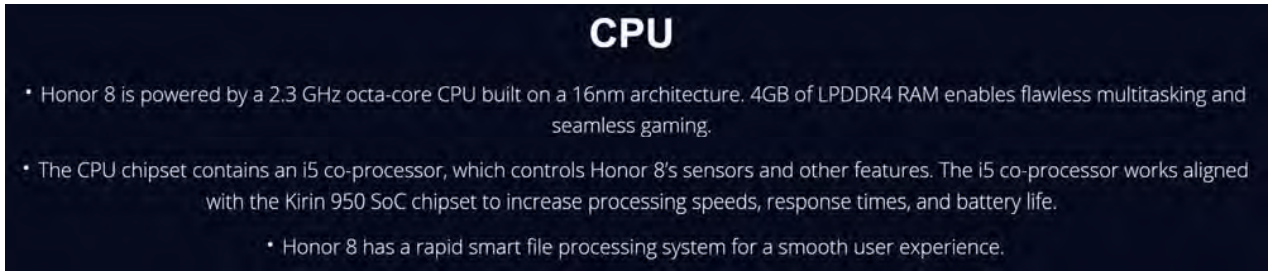
[iv] a microprocessor adapted to:	The Huawei Honor 8 includes at least one microprocessor. 

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	See https://www.hihonor.com/global/products/smartphone/honor8/, *last accessed December 5, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 8 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Huawei Honor 8 is an Android phone. The Huawei Honor 8's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `  public OutgoingReceiver() {` `  }` `  @Override` `  public void onReceive(Context context, Intent intent) {` `  String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `  Toast.makeText(ctx,` `  "Outgoing: "+number,` `  Toast.LENGTH_LONG).show();` `  }` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`. Incoming Call:

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	`` /** * Listener to detect incoming calls. */ private class CallStateListener extends PhoneStateListener { @Override public void onCallStateChanged(int state, String incomingNumber) { switch (state) { case TelephonyManager.CALL_STATE_RINGING: // called when someone is ringing to this phone Toast.makeText(ctx, "Incoming: "+incomingNumber, Toast.LENGTH_LONG).show(); break; } } } ``` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Honor 8 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

508

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

509

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor 8, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Honor 8 is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

510

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

511

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

512	**Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.*

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

513

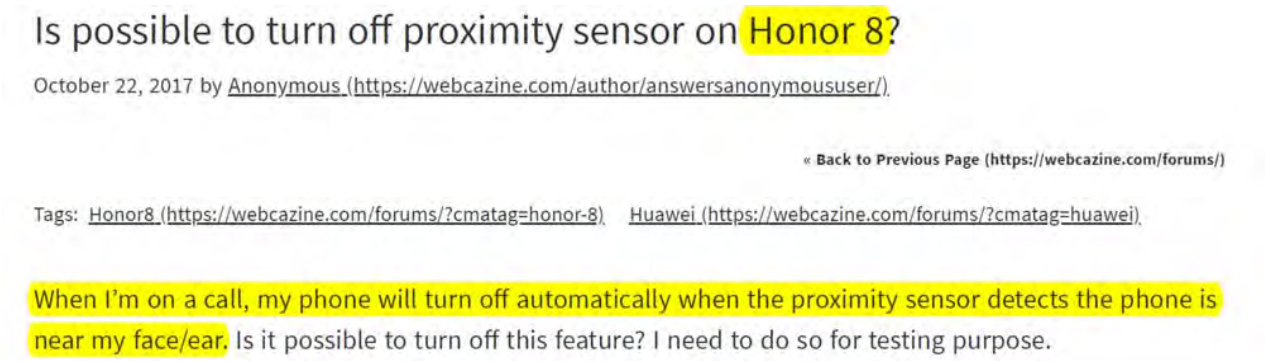
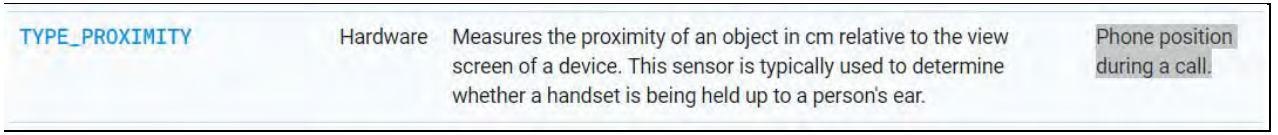
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Honor 8 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists. By way of example only, the following excerpted document shows that when the Huawei Honor 8 is proximate to an object, and a call has been initiated or answered, power to the display is reduced and the display is darkened.  Is possible to turn off proximity sensor on Honor 8? October 22, 2017 by Anonymous (https://webcazine.com/author/answersanonymoususer/) « Back to Previous Page (https://webcazine.com/forums/) Tags: Honor8 (https://webcazine.com/forums/?cmatag=honor-8) Huawei (https://webcazine.com/forums/?cmatag=huawei) When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose. *See* https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/, *last accessed December 5, 2018.* By way of further example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:  TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

514

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `private lateinit var mSensorManager: SensorManager` `private var mProximity: Sensor? = null` `public override fun onCreate(savedInstanceState: Bundle?) {` `super.onCreate(savedInstanceState)` `setContentView(R.layout.main)` `// Get an instance of the sensor service, and use that to get an instance of` `// a particular sensor.` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `}` `override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `// Do something here if sensor accuracy changes.` `}`

515

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	516 `override fun onSensorChanged(event: SensorEvent) {` `  val distance = event.values[0]` `  // Do something with this sensor data.` `}` `override fun onResume() {` `  // Register a listener for the sensor.` `  super.onResume()` `  mProximity?.also { proximity ->` `  mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)` `  }` `}` `override fun onPause() {` `  // Be sure to unregister the sensor when the activity pauses.` `  super.onPause()` `  mSensorManager.unregisterListener(this)` `}` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Honor 8 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Honor 8, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Honor 8 reduces power to the display by turning off the display. By way of example only, the following excerpted document shows the Huawei Honor 8 reducing power and turning off the screen. Is possible to turn off proximity sensor on Honor 8? October 22, 2017 by Anonymous (https://webcazine.com/author/answersanonymoususer/) « Back to Previous Page (https://webcazine.com/forums/) Tags: Honor8 (https://webcazine.com/forums/?cmatag=honor-8) Huawei (https://webcazine.com/forums/?cmatag=huawei) When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

518

See https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/, *last accessed December 5, 2018.*

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

519

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:	

520

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

521	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Honor 8 is a range-detecting sensor. The proximity sensor detects the range of a proximate object. By way of example only, when an incoming call is answered, the mobile station's display is on or off depending on the range of a proximate object.

522

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

523

Is possible to turn off proximity sensor on Honor 8?

October 22, 2017 by Anonymous (https://webcazine.com/author/answersanonymoususer/)

« Back to Previous Page (https://webcazine.com/forums/)

Tags: Honor8 (https://webcazine.com/forums/?cmatag=honor-8) Huawei (https://webcazine.com/forums/?cmatag=huawei)

When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.

See https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/, *last accessed December 5, 2018.*

By way of further example only, the Android Developer Code Website describes range detection in proximity sensors:

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	The Android Developer Code Website further explains: **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method. *See id.*

524

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Honor 8 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call. By way of example only, the following excerpted document suggests the Huawei Honor 8 reduces power and darkens the display due to object proximity substantially concurrently with an initiated wireless telephone call. Is possible to turn off proximity sensor on Honor 8? October 22, 2017 by Anonymous (https://webcazine.com/author/answersanonymoususer/) « Back to Previous Page (https://webcazine.com/forums/) Tags: Honor8 (https://webcazine.com/forums/?cmatag=honor-8) Huawei (https://webcazine.com/forums/?cmatag=huawei) When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose. *See* https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/, *last accessed December 5, 2018.* By way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

525

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

526	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

527	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Honor 8 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

528

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Honor 8 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Honor 8 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

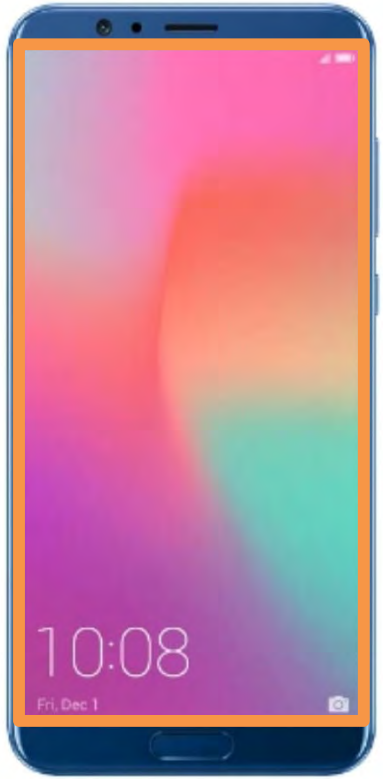
and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 8 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).

530

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

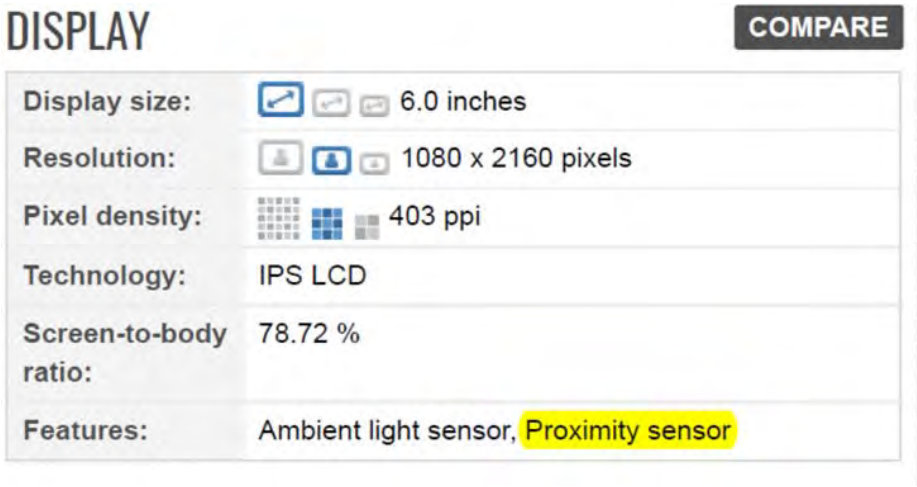
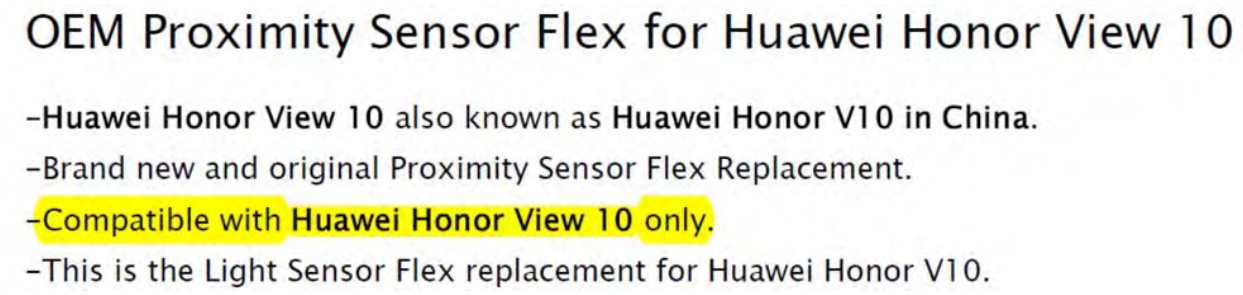
Asserted Claim Elements	Huawei Honor View10
1. A mobile station, comprising:	To the extent that the preamble is found to be limiting, the Huawei Honor View10 is a mobile station.  *See* https://www.hihonor.com/us/product/10001859603314.html#1000000100132, *last accessed December 5, 2018.*

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

[i] a display;	The Huawei Honor View10 includes a display, outlined in the figure below.  *See* https://www.hihonor.com/us/product/10001859603314.html#1000000100132, *last accessed December 5, 2018.*

532

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

[ii] a proximity sensor	The Huawei Honor View10 includes a proximity sensor.  *See* https://www.phonearena.com/phones/Honor-View-10_id10752, last accessed December 6, 2018. See also OEM replacement part for proximity sensor compatible only with Huawei View 10: 

533

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	See https://www.witrigs.com/oem-proximity-sensor-flex-for-huawei-honor-view-10, last accessed December 6, 2018.
[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and	The proximity sensor of the Huawei Honor View10 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate . By way of example only, the Huawei Honor View10 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor. TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See SensorEvent.values for more details. **See also:** isWakeUpSensor() Constant Value: 8 (0x00000008)

534

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

535	*See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.* The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals: isWakeUpSensor added in API level 21 `public boolean isWakeUpSensor ()` Returns true if the sensor is a wake-up sensor. **Application Processor Power modes** Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more. **Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	person's head is from the face of a handset device (for example, when a user making or receiving a phone call)." Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

536

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

537	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
[iv] a microprocessor adapted to:	The Huawei Honor View10 includes at least one microprocessor. SPECIFICATIONS CHIPSET 4*Cortex A73 2.36GHz + 4*Cortex A53 1.8GHz

538

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	See https://www.hihonor.com/us/product/10001859603314.html#1000000100132 , *last accessed December 5, 2018.*
(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor View 10 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. By way of example only, the Huawei Honor View 10 is an Android phone. The Huawei Honor View 10's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call; Outgoing Call: `/**` `* Broadcast receiver to detect the outgoing calls.` `*/` `public class OutgoingReceiver extends BroadcastReceiver {` `public OutgoingReceiver() {` `}` `@Override` `public void onReceive(Context context, Intent intent) {` `String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);` `Toast.makeText(ctx,` `"Outgoing: "+number,` `Toast.LENGTH_LONG).show();` `}` `}` In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

539

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	Incoming Call: `/**` `* Listener to detect incoming calls.` `*/` `private class CallStateListener extends PhoneStateListener {` `  @Override` `  public void onCallStateChanged(int state, String incomingNumber) {` `  switch (state) {` `  case TelephonyManager.CALL_STATE_RINGING:` `  // called when someone is ringing to this phone` `  Toast.makeText(ctx,` `  "Incoming: "+incomingNumber,` `  Toast.LENGTH_LONG).show();` `  break;` `  }` `  }` `}` In the above, `state` is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call. *See* https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, *last accessed December 5, 2018.*
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	The microprocessor of the Huawei Honor View10 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor. By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition. The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

541

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

542

	Non-wake-up sensors Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor View10, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.
(c) receive the signal from the activated proximity sensor, and	The microprocessor of the Huawei Honor View10 is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

543

TYPE_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

544	**Non-wake-up sensors** Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually: • Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case. • Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost. **Wake-up sensors** In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details. *See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.* In the Huawei Honor View 10, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	The microprocessor of the Huawei Honor View10 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists . By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate: TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call. *See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.* In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (or example, when a user making or receiving a phone call)."

545

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

546	Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

547	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

548

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
2. The mobile station of claim 1,	*See* claim 1.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.	The Huawei Honor View10 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists. In Huawei Honor View10, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased. By way of example, the Android Developer Code Website describes that "[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call)." *See* *https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox*, *last accessed November 30, 2018.*
4. The mobile station as recited in claim 1,	*See* claim 1.
wherein the microprocessor reduces power to the display by turning off the display.	The microprocessor of the Huawei Honor View10 reduces power to the display by turning off the display. By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

549

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

550

TYPE_PROXIMITY

`public static final int TYPE_PROXIMITY`

A constant describing a proximity sensor type. This is a wake up sensor.

See `SensorEvent.values` for more details.

See also:

`isWakeUpSensor()`

Constant Value: 8 (0x00000008)

See https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY, last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor — added in API level 21

`public boolean isWakeUpSensor ()`

Returns true if the sensor is a wake-up sensor.

Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

551

Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor(), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

552

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

KOTLIN JAVA

```
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

553	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
5. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.	The proximity sensor of the Huawei Honor View10 is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

555	

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See https://developer.android.com/guide/topics/sensors/sensors_overview, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}
```

See https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, last accessed November 29, 2018.

The Android Developer Code Website further explains:

> **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See id.

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

7. The mobile station as recited in claim 1,	*See* claim 1.
wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.	The proximity sensor of the Huawei Honor View10 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call . By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code: Use the proximity sensor The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor: KOTLIN JAVA `private lateinit var mSensorManager: SensorManager` `private var mSensor: Sensor? = null` `...` `mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

556

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

557	KOTLIN JAVA `class SensorActivity : Activity(), SensorEventListener {` `  private lateinit var mSensorManager: SensorManager` `  private var mProximity: Sensor? = null` `  public override fun onCreate(savedInstanceState: Bundle?) {` `  super.onCreate(savedInstanceState)` `  setContentView(R.layout.main)` `  // Get an instance of the sensor service, and use that to get an instance of` `  // a particular sensor.` `  mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager` `  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)` `  }` `  override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {` `  // Do something here if sensor accuracy changes.` `  }`

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	`` override fun onSensorChanged(event: SensorEvent) { val distance = event.values[0] // Do something with this sensor data. } override fun onResume() { // Register a listener for the sensor. super.onResume() mProximity?.also { proximity -> mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL) } } override fun onPause() { // Be sure to unregister the sensor when the activity pauses. super.onPause() mSensorManager.unregisterListener(this) } ``` *See* https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox, *last accessed November 29, 2018.*
8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:	To the extent that the preamble is found to be limiting, *see* claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).
the mobile station detecting the existence of an initiated call condition or an answered-call	The Huawei Honor View10 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being

558

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;	that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call. *See* claim 1(a).
the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and	The Huawei Honor View10 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. *See* claim 1(b).
the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.	The Huawei Honor View10 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. *See* claim 1(c)-(d).
14. A mobile station, comprising:	*See* claim 1(preamble).
a display;	*See* claim 1[i].
a proximity sensor	*See* claim 1[ii].
adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;	*See* claim 1[iii].

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

and a microprocessor adapted to:	*See* claim[iv].
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor View10 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call. *See* claim 1(a).
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	*See* claim 1(b).
(c) receive the signal from the activated proximity sensor, and	*See* claim 1(c).
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	*See* claim 1(d).