

DOCKET NO.: 0107131.00562US5

Filed on behalf of Intel Corp.

By: Jason Kipnis, Reg. No. 40,680

David Cavanaugh, Reg. No. 36,476

Joseph F. Haag, Reg. No. 42,612

Wilmer Cutler Pickering Hale and Dorr LLP

950 Page Mill Road

Palo Alto, CA 94304

Tel: (650) 858-6000

Email: Jason.Kipnis@wilmerhale.com

UNITED STATES PATENT AND TRADEMARK OFFICE

---

**BEFORE THE PATENT TRIAL AND APPEAL BOARD**

---

Intel Corporation  
Petitioner

v.

Qualcomm Incorporated  
Patent Owner

---

Case IPR2018-01346

---

**PETITION FOR *INTER PARTES* REVIEW OF  
U.S. PATENT NO. 9,535,490  
CHALLENGING CLAIMS 9, 11-13, 26, AND 27**

## TABLE OF CONTENTS

I.	INTRODUCTION .....	1
II.	MANDATORY NOTICES .....	1
A.	Real Party-in-Interest .....	1
B.	Related Matters.....	1
C.	Counsel .....	2
D.	Service Information.....	2
III.	CERTIFICATION OF GROUNDS FOR STANDING .....	3
IV.	OVERVIEW OF CHALLENGE AND RELIEF REQUESTED.....	3
A.	Prior Art Patents and Printed Publications.....	3
B.	Grounds for Challenge .....	4
V.	TECHNOLOGY BACKGROUND.....	4
A.	Processor-To-Processor Communications .....	4
B.	Communication Bus Power-Saving States.....	6
C.	Data Buffering .....	7
VI.	THE '490 PATENT .....	9
A.	Alleged Problem of the Prior Art .....	10
B.	Purported Solution of the '490 Patent .....	11
C.	Prosecution History of the '490 Patent .....	16
VII.	CLAIM CONSTRUCTION .....	18
A.	“triggered by” .....	19

VIII.	OVERVIEW OF PRINCIPAL PRIOR ART REFERENCES .....	20
A.	U.S. Patent No. 9,329,671 To Heinrich et al.....	20
B.	U.S. Patent No. 8,160,000 to Balasubramanian .....	23
IX.	SPECIFIC GROUNDS FOR PETITION .....	27
A.	Ground 1: Claims 9, 11-13, 26, and 27 Are Rendered Obvious By Heinrich In View Of Balasubramanian .....	27
1.	Claim 1 .....	28
2.	Claim 9 .....	55
3.	Claim 12 .....	57
4.	Claim 11 .....	59
5.	Claim 13 .....	61
6.	Claim 26 .....	64
7.	Claim 27 .....	67
X.	CONCLUSION.....	69

## **I. INTRODUCTION**

Intel Corporation (“Petitioner”) respectfully submits this Petition for *Inter Partes* Review (“Petition”) of claims 9, 11-13, 26, and 27 of U.S. Patent No. 9,535,490 (the “’490 patent”) (Ex-1401). The ’490 patent discloses a device comprising two processors that communicate over a bus using a trigger protocol. *See* Ex-1401, claims 9, 11-13, 26, and 27. However, there was nothing inventive about the claimed device as of the earliest priority date of the ’490 patent, and the claimed concepts had been well-known long before the ’490 patent’s earliest priority date. Thus, claims 9, 11-13, 26, and 27 of the ’490 patent should be canceled.

## **II. MANDATORY NOTICES**

### **A. Real Party-in-Interest**

Intel Corporation is the real party-in-interest and submits this *inter partes* review petition for review of certain claims of U.S. Patent No. 9,535,490. Petitioner also identifies Apple Inc. (“Apple”) as a real party-in-interest.

### **B. Related Matters**

Qualcomm Incorporated (“Qualcomm” or “Patent Owner”) has asserted the ’490 patent against Apple in *Certain Mobile Elec. Devices and Radio Frequency Components Thereof*, Inv. No. 337-TA-1065 (Int’l Trade Comm’n) currently pending before the International Trade Commission. Qualcomm also has

asserted the '490 patent against Apple in *Qualcomm Inc. v. Apple. Inc.*, Case No. 3:17-cv-01375-DMS-MDD (S.D. Cal.).

Concurrently with this *inter partes* review petition, Petitioner is also filing *inter partes* review petitions for claims 20, 28, and 30 of the '490 patent (IPR2018-01344). Petitioner has also previously filed *inter partes* review petitions for claims 1-6 and 8 of the '490 patent (IPR2018-01293), claim 31 of the '490 patent (IPR2018-01261), and claims 16-17 and 22-24 of the '490 patent (IPR2018-01295). Petitioner requests that these petitions be assigned to the same panel.

**C. Counsel**

Lead Counsel: Jason Kipnis (Registration No. 40,680)

Backup Counsel: David L. Cavanaugh (Registration No. 36,476)

Backup Counsel: Joseph F. Haag (Registration No. 42,612)

Petitioner also plans to file *pro hac vice* applications for Joseph Mueller, Nina Tallon, and Todd Zubler, each counsel of record in the pending litigation.

**D. Service Information**

Email: Jason.Kipnis@wilmerhale.com; David.Cavanaugh@wilmerhale.com; Joseph.Haag@wilmerhale.com.

Post and hand delivery: Wilmer Cutler Pickering Hale and Dorr LLP

950 Page Mill Road

Palo Alto, CA 94304

Telephone: 650-858-6000

Facsimile: 650-858-6100

Petitioner consents to service by email.

### **III. CERTIFICATION OF GROUNDS FOR STANDING**

Petitioner certifies pursuant to Rule 42.104(a) that the patent for which review is sought is available for *inter partes* review and that Petitioner is not barred or estopped from requesting an *inter partes* review challenging the patent claims on the grounds identified in this Petition.

### **IV. OVERVIEW OF CHALLENGE AND RELIEF REQUESTED**

Pursuant to Rules 42.22(a)(1) and 42.104(b)(1)-(2), Petitioner challenges claims 9, 11-13, 26, and 27 of the '490 patent (Ex-1401).

#### **A. Prior Art Patents and Printed Publications**

Petitioner relies upon the following patents and printed publications:

1. U.S. Patent No. 9,329,671 to Heinrich et al. ("Heinrich") (Ex-1404), which was filed January 29, 2013 and issued May 3, 2016, is prior art to the '490 patent at least under post-AIA 35 U.S.C. § 102(a)(2).
2. U.S. Patent No. 8,160,000 to Balasubramanian ("Balasubramanian") (Ex-1405), which was filed October 3, 2006 and issued April 17, 2012, is prior art to the '490 patent at least under post-AIA 35 U.S.C. § 102(a)(1).

## **B. Grounds for Challenge**

Petitioner requests cancellation of claims 9, 11-13, 26, and 27 as unpatentable under 35 U.S.C. § 103. This Petition, which is supported by the Declaration of Dr. Bill Lin (“Lin”) (Ex-1402), demonstrates a reasonable likelihood that Petitioner will prevail with respect to at least one challenged claim and that each challenged claim is unpatentable for the reasons cited herein. *See* 35 U.S.C. § 314(a).

The grounds for challenge based on the foregoing prior art references include the following:

	<b>Grounds</b>	<b>Reference(s)</b>	<b>Challenged Claims</b>
1.	§ 103	Combination of Heinrich and Balasubramanian	9, 11-13, 26, and 27

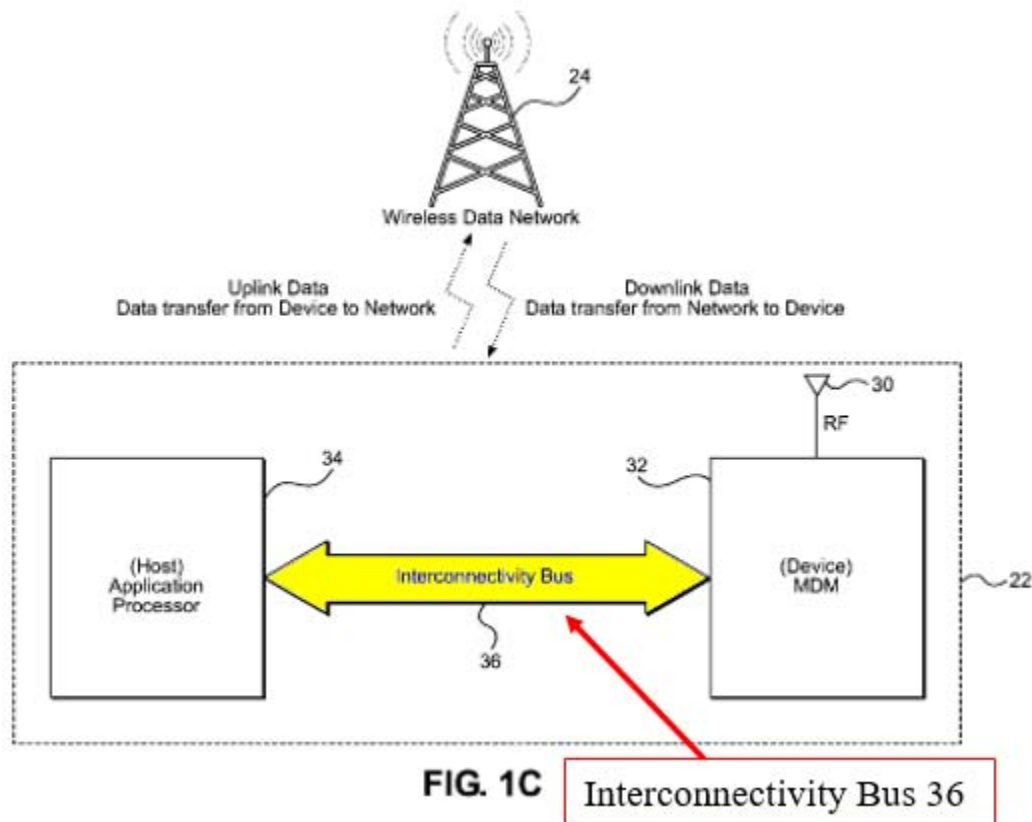
## **V. TECHNOLOGY BACKGROUND**

### **A. Processor-To-Processor Communications**

The '490 patent generally relates to communications between two processing nodes within a computing device—(1) a modem processor, which typically manages the transmission and reception of data over a network (*e.g.*, over a cellular or Wi-Fi network) and (2) an application processor, which typically runs applications on the device (*e.g.*, email, text messaging, and web browsing programs). Lin-¶28.

For example, when a mobile device user composes a text message using a software application running on the device's application processor, the application processor transmits the text data to a modem processor. After processing the data to allow it to be transmitted wirelessly, the modem processor manages transmission of the data to the relevant network. Similarly, when a network transmits data to the mobile device (*e.g.*, data for a voice call or a requested web page), the modem processor processes the incoming data and then sends it to the application processor. The relevant application on the application processor can use the data (*e.g.*, a phone application can play received voice data or a web browser application can display received web page data). Lin-¶29.

These processor-to-processor communications typically occur over a wire or set of wires commonly known as a communication "bus." For instance, Figure 1C of the '490 patent below shows application processor 34 and mobile device modem ("MDM") 32 connected by interconnectivity bus 36:



Ex-1401, Fig. 1C (annotations added). As shown in the figure, the '490 patent refers to data sent from the application processor to the modem processor and then to the wireless data network as “uplink data”, and it refers to data sent from that network to the modem processor and then to the application processor as “downlink” data. Lin-¶30.

## B. Communication Bus Power-Saving States

Like other electronic components, a communication bus must be powered for electrical signals to flow across it. But maintaining a bus in an “active” state consumes power. Therefore, buses are often designed to have one or more power-saving (“low power”) states, during which the bus or components attached to the

bus are powered down (partially or fully) such that data cannot flow across the bus. Ex-1401, 8:6-19 (“In conventional mobile terminals that have a PCIe interconnectivity bus (i.e., the interconnectivity bus 36), the PCIe standard allows the interconnectivity bus 36 to be placed into a sleep mode. . . . This problem is not unique to the PCIe interconnectivity bus 36.”); Lin-¶31.

If a processor has data to transmit to another processor, the data can be sent right away if the bus connecting them is already in an active state. However, if the bus is in a low power state at the time, the bus must transition to the active state before the processor can send the data. As the ’490 patent notes, these bus transitions themselves require power. Ex-1401, 8:9-12 (“While placing the interconnectivity bus 36 in a sleep mode generally saves power, such sleep modes do have a drawback in that they consume relatively large amounts of power as they transition out of the sleep mode.”); Lin-¶32.

### **C. Data Buffering**

A first processor may have data to send to a second processor, but the communication bus between them, or the second processor itself, may be inactive, thus preventing the transmission of the data. The first processor could wake the bus whenever it has data to send, but frequent transitions of the bus from a low power state to a high power state can waste power. The first processor could simply discard the data—but that would result in lost or incomplete data. Lin-¶33.

Given these obvious drawbacks, it has long been known that processor-to-processor data can be held in a “buffer” (a temporary, short-term memory) during periods when the bus (or receiving processor) is in a low power state. Lin-¶34; *see, e.g.*, Ex-1405, 5:47-51, 6:40-43, 9:4-7; Ex-1404, 8:21-64. Collecting multiple data packets over time and sending them together after the bus transitions to an active state—rather than waking the bus from a low power state each time the processor receives a data packet—saves power by reducing the number of bus power state transitions. Lin-¶34.

Buffers, however, have limitations. Once a buffer is filled with data, additional incoming packets cannot be stored. Moreover, if a buffer stores data for too long, the data may become too old to be useful (*i.e.*, stale), or the delay may negatively affect applications expecting to receive the data. For example, in a real-time phone conversation, if voice data is held in a buffer too long, users can experience unpleasant gaps in communication. Therefore, a processor must typically send buffered data before the buffer fills up and before the data becomes stale. Lin-¶35.

The prior art describes many ways to determine when to send buffered data. One example is a “timer,” which tracks how long data has been held in a buffer; when the timer expires, the buffered data is sent. *See, e.g.*, Ex-1404, 9:22-24; Ex-1405, 1:66-2:2, 6:55-65, 9:7-9. Another example is a “counter,” which can track

the number of data packets or bytes held in the buffer or count a number of specified events that occur. *See, e.g.*, Ex-1405, 2:3-8, 6:55-65, 9:7-9. When the counter reaches a predefined threshold (*e.g.*, a predefined number of packets, data, or events), the buffered data is sent. *Id.* Such timers and counters can be used together to ensure that (1) the buffer does not hold data for too long and (2) the buffer does not fill up. Lin-¶36.

## **VI. THE '490 PATENT**

The application leading to the '490 patent (Ex-1401) was filed as U.S. Application No. 14/568,694 on December 12, 2014. The '490 patent claims priority to U.S. Provisional Application No. 61/916,498, filed December 16, 2013, and U.S. Provisional Application No. 62/019,073, filed June 30, 2014.<sup>1</sup>

The '490 patent is directed to systems and methods that claim to conserve power by limiting the number of transitions of a device between an active state and a low power state.

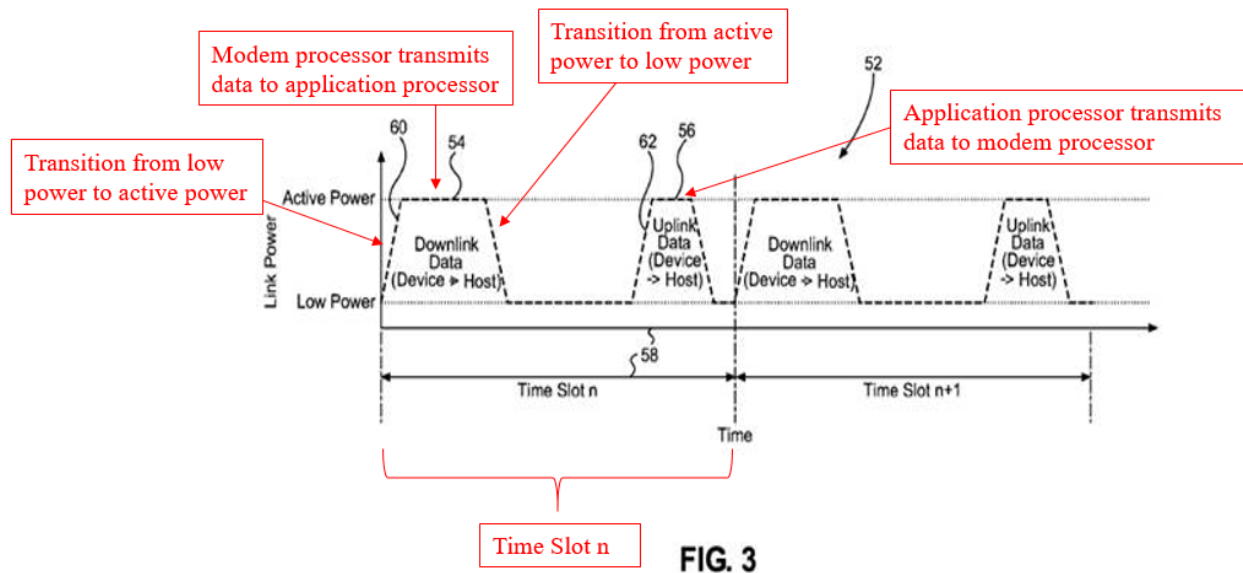
---

<sup>1</sup> For purposes of this Petition, Petitioner treats December 16, 2013 as the effective filing date, but does not take any position regarding whether the claims in the '490 patent are enabled by or have written description support in the '498 and/or '073 provisional applications.

**A. Alleged Problem of the Prior Art**

The '490 patent admits that, for prior art mobile devices containing an application processor and modem processor coupled via a communication bus, it was known that power-savings could be achieved by putting the bus into a low power state during certain periods of time. Ex-1401, 8:6-10 (“In conventional mobile terminals that have a PCIe interconnectivity bus . . . , the PCIe standard allows the interconnectivity bus 36 to be placed into a sleep mode. . . . [P]lacing the interconnectivity bus 36 in a sleep mode generally saves power, . . .”).

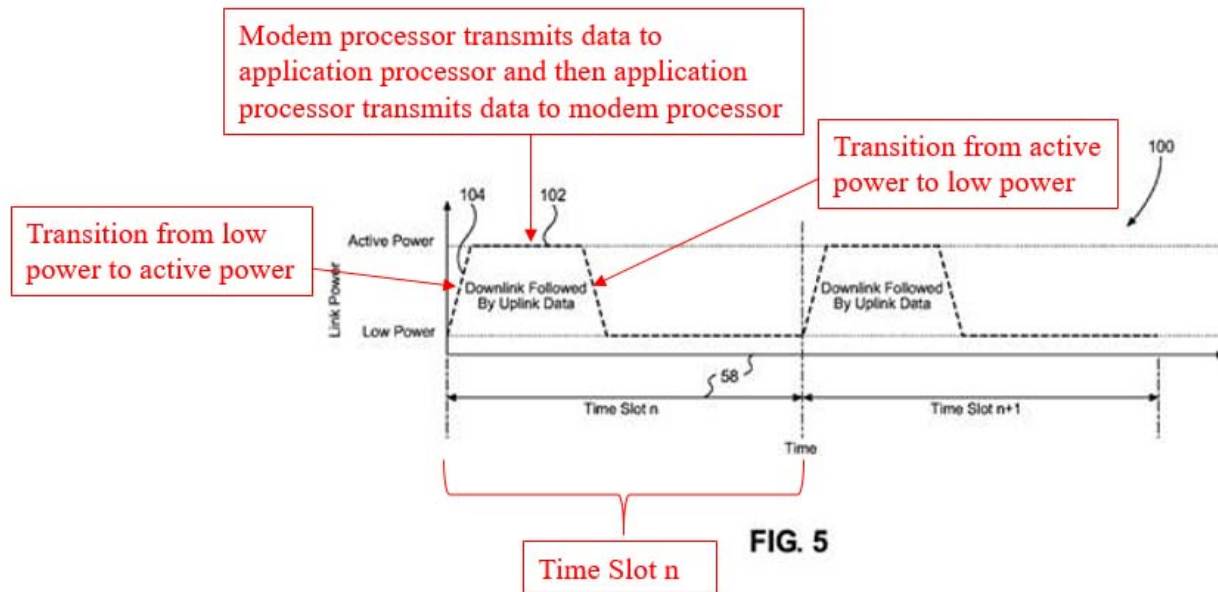
According to the '490 patent, however, these prior art devices wasted power by transitioning power states too frequently, because they: (1) transitioned the bus from a low power state to an active state to transmit downlink data, and (2) separately performed the same bus transition again at a later time to transmit uplink data—as shown in Figure 3 below:



Ex-1401, Fig. 3 (annotations added); *id.*, 8:35-40 (stating that “two transitions (i.e., 60, 62) from low power to active power [] each time slot 58 . . . [will] consume substantial amounts of power and reduce the battery life of the mobile terminal 22”); *id.*, 8:10-12 (“[S]uch sleep modes do have a drawback in that they consume relatively large amounts of power as they transition out of the sleep mode.”); Lin-¶39.

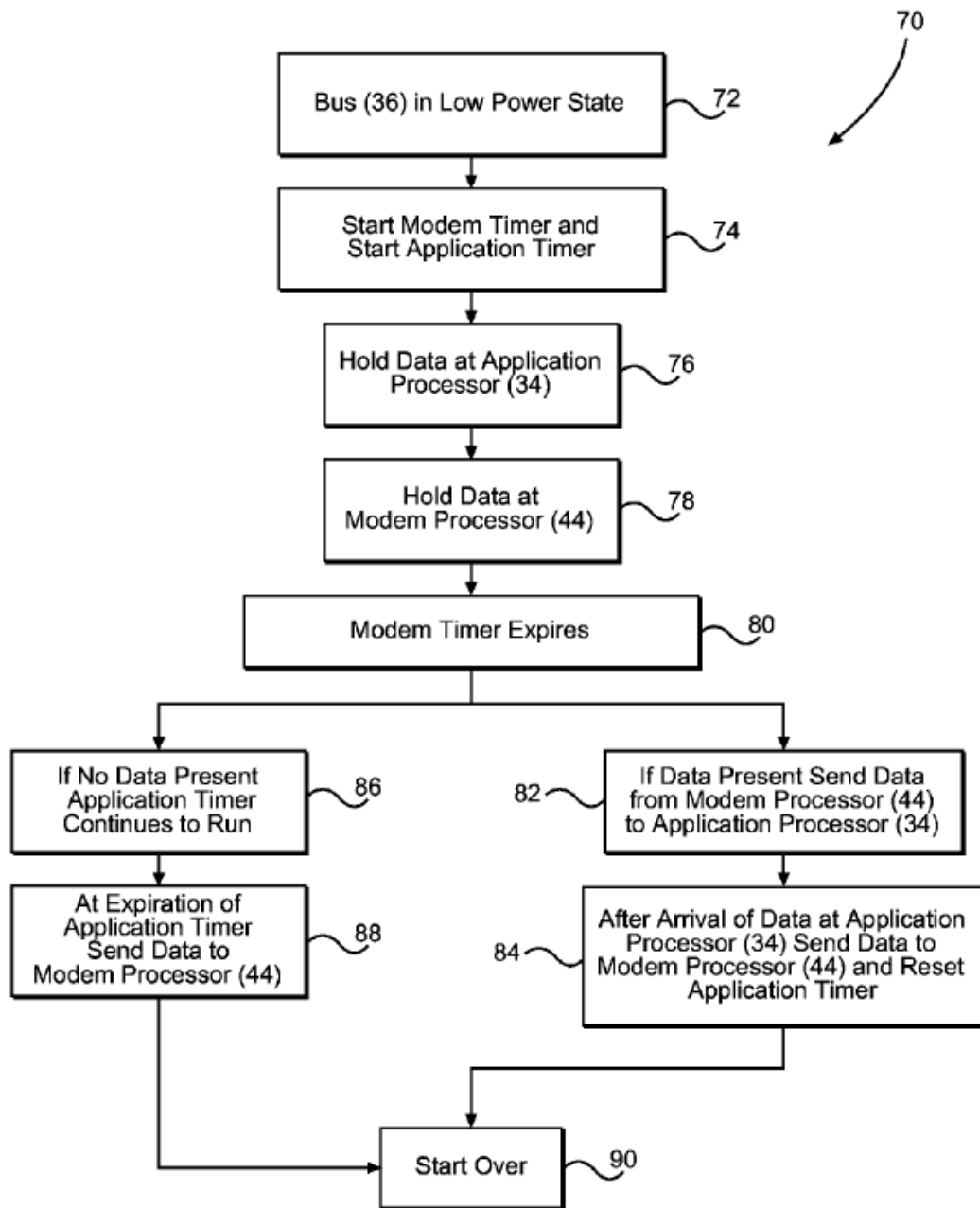
## B. Purported Solution of the ’490 Patent

To address this supposed “problem,” the ’490 patent does not claim to invent a new type of processor, a new type of communication bus, or a new type of bus power-saving state. Instead, the patent claims to improve power-savings merely by transmitting buffered downlink data followed by buffered uplink data during the *same active period of the communications bus*—as shown in Figure 5 below:



Ex-1401, Fig. 5 (annotations added); *id.*, 10:40-45 (“Thus, by consolidating the data into a single active period 102, the overall time that is spent in low power may be increased, thus resulting in power savings. Additionally, power spent transitioning from a low power to active power state is reduced by elimination of the second transition 62.”); Lin-¶40.

The ’490 patent discusses an “exemplary” embodiment using this single bus transition scheme, as shown below in Figure 4:



**FIG. 4**

Ex-1401, Fig. 4. At step 72 the bus is in a “low power” state during which data cannot be transmitted over the bus. *Id.*, 9:22-23. At step 74, a timer starts at both the modem processor and application processor. *Id.*, 9:23-27. While the bus is

inactive and the timers are running, the application processor 34 holds any data that applications generate for sending to the modem processor (step 76), and the modem processor 44 holds any data that it receives from the network for sending to the application processor (step 78). *Id.*, 9:27-32 (“Data is generated by the application processor 34 and data is received from the network 12 by the modem processor 44. The application data is held at the application processor 34 (block 76), and the modem data is held at modem processor 44 (block 78) while the timers are running.”); Lin-¶41.

When the modem timer expires at step 80, if the modem processor has buffered data, the bus transitions from a low power state to an active state—after which modem processor 44 sends its buffered data to application processor 34 over the communication bus (at step 82). Ex. 1401, 9:37-40. After receiving all the buffered data from the modem processor, the application processor treats the receipt of that data as a “trigger” that causes the application processor to send any data that it has buffered to modem processor 44 before the bus transitions back to a low power state (at step 84). *Id.*<sup>2</sup>; Lin-¶42.

---

<sup>2</sup> For steps 86 and 88 of Figure 4, the patent explains that if the modem processor has not buffered any data when the modem timer expires, the application

As detailed further below, transmitting all accumulated downlink and uplink data during the same active state was known—long before the claimed priority date of the '490 patent—as a common sense and predictable way to reduce the power consumed by a computing system. *See, e.g.*, Ex-1405, 6:63-7:6 (“[T]he transceiver 110 ***then transmits the queued uplink packets over the communication link 116.*** Advantageously, the queued packets may be grouped for transmission such that ***all of the packets are transmitted during a single wake state of the transceiver 110.*** For example, as discussed above the transceiver 110 may send the queued packets in relative close succession (e.g., back-to-back) over the communication link. As represented by block 210, during the ***same single wake state the transceiver 110 also receives any downlink packets queued in the network interface 112.***” (emphasis added)); Lin-¶44.

Also well-known was the specific scheme disclosed in the '490 patent for how to transmit all buffered downlink and uplink data during the same active state: namely, using the receipt of buffered data from a first processing node as a “trigger” for a second processing node to send its buffered data. For example, nearly a decade before the '490 patent was filed, Balasubramanian disclosed a

---

processor will send any data that it has buffered to the modem processor when the application timer later expires. Ex-1401, 10:4-10.

system in which receipt of data from a first processing node “trigger[ed]” the transmission of data from a second processing node. Ex-1405, 7:6-8 (“For example, the network interface 112 may use the *receipt* of an uplink packet as a *trigger* to transmit any downlink packets in its queue.”) (emphasis added); Lin-¶45.

### C. Prosecution History of the ’490 Patent

U.S. Application No. 14/568,694 (“the ’694 application”), which issued as the ’490 patent, was filed on December 12, 2014. The original application was filed with 29 claims, including 9 independent claims. On April 27, 2015, the Applicant added two more claims via preliminary amendment. Ex-1403, [Apr. 27, 2015 Preliminary Amendment], 8.

Initially, all claims of the ’490 patent were rejected over the prior art. The Examiner allowed the application only after the Applicant amended most of the claims to include the “trigger” limitation—which requires a second processor to be triggered to send held data to a first processor in response to receiving data from the first processor. In particular, on June 10, 2016, the Examiner rejected all pending claims over PCT Publication No. WO 2009/039034 (“the Intel PCT”) (Ex-1406) and U.S. Patent No. 6,021,264 (“Morita”) (Ex-1407). *See* Ex-1403, [6/10/2016 Office Action], 3-9. In response to this rejection, the Applicant amended claim 1 to include the “trigger” limitation as shown below:

1. A mobile terminal comprising:

a modem timer;

a modem processor, the modem processor configured to hold modem processor to application processor data until expiration of the modem timer;

an application processor;

an interconnectivity bus communicatively coupling the application processor to the modem processor; and

the application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.

Ex-1403 [8/24/2016 Response], 2. The Applicant similarly amended original independent claims 15, 20, and 24-28 to include a “trigger” limitation. *Id.*, 14 (“Independent claims 15, 20, and 24-29 have been amended to recite similar features and are therefore allowable for at least the same reasons as claim 1, . . .”); Lin-¶47.

The Applicant argued that Morita does not disclose that the transmission of data from the application processor to the modem processor is “triggered by” receipt of data from the modem processor. Ex-1403, 14. Instead, the Applicant argued, Morita discloses “delay[ing]” transmission of data “by a predetermined time length.” *Id.* As a result, the Applicant argued, that transmission could not be “triggered by” or “responsive to” receipt of data from the modem processor because the data is held for a “predetermined time length,” which, by its definition, is a length of time determined in advance of receiving any data. *Id.* The Applicant did not separately argue that the Intel PCT fails to disclose any of the other limitations. *Id.*, 13-15; Lin-¶49.

The Examiner subsequently allowed the claims as amended.<sup>3</sup>

## VII. CLAIM CONSTRUCTION

Petitioner has set forth below its proposed constructions of certain terms of the '490 patent and its support for the constructions. 37 C.F.R. § 42.100(b) states that claims must be given their broadest reasonable construction in light of the specification (“BRI standard”). On May 8, 2018, the USPTO proposed rulemaking that would change the standard for construing claims from the BRI standard to the

---

<sup>3</sup> The claims were re-ordered such that original claims 15, 20, and 24-29 became issued claims 16, 22, and 26-31, respectively.

Phillips standard. In anticipation that the rule change will apply to these proceedings, Petitioner construes the claims based on the standard set forth in Phillips. Petitioner is not aware of any difference in how the claims would be construed under the BRI standard. The scope of the challenged claims could not be broader under the proposed Phillips construction than it could be under the BRI standard. Therefore, the challenged claims would also be unpatentable under the BRI standard.

A person of ordinary skill in the art (“POSA”) of the ’490 patent would have had a Master’s degree in Electrical Engineering, Computer Engineering, or Computer Science plus at least two years of experience in mobile device architecture and multiprocessor systems, or alternatively a Bachelor’s degree in one of those fields plus at least four years of experience in mobile device architecture and multiprocessor systems. Lin-¶70.

**A. “triggered by”**

As used in the ’490 patent, a POSA would have understood the phrase “triggered by” to mean “initiated in response to”. The ’490 patent does not specifically define the term “trigger,” but implies that “triggering” occurs when an action is taken in response to a stimulus. *See* Ex-1401, 2:3-6 (“On receipt of the data from the modem processor, the application processor sends data held by the application processor to the modem processor over the PCIe interconnectivity

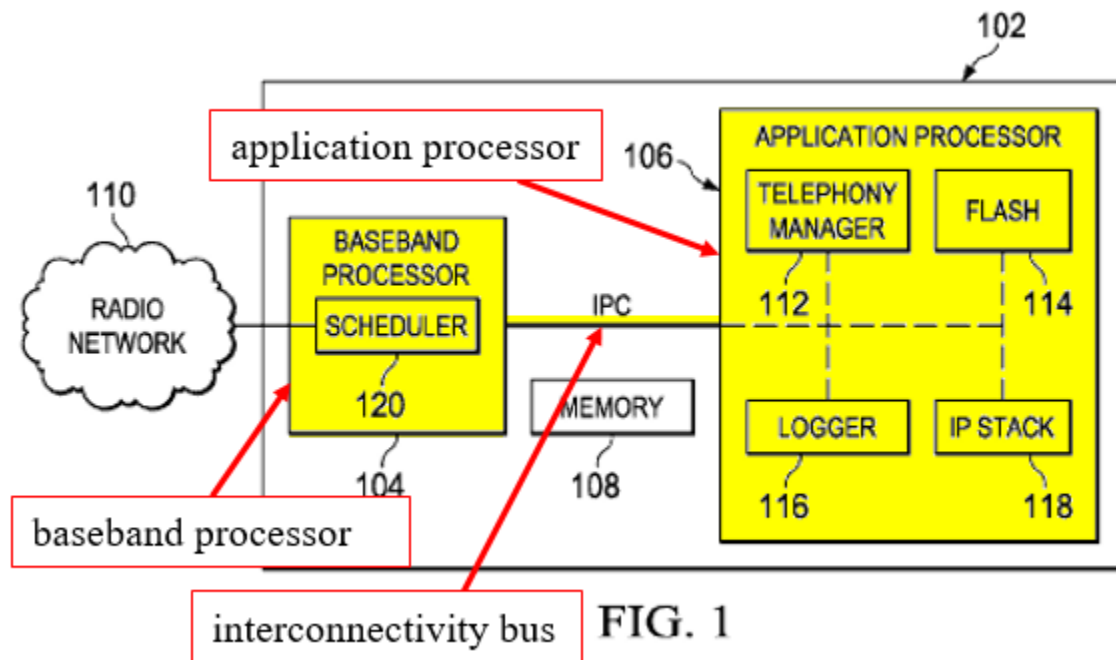
bus.”); *see also id.*, 11:15-18, 11:39-46. This understanding of “trigger” is supported by the claims themselves. For example, claim 1 states that the application processor is “triggered by [the] receipt of” data from the modem processor, and suggests that this is synonymous with being “responsive to the receipt of” data from the modem processor. *See id.*, claim 1. The prosecution history also supports this understanding of “triggered by.” *See* Ex-1403 [8/24/2016 Response], 14 (“[T]here is no disclosure or suggestion that the ‘predetermined time length’ is even capable of being ‘triggered by’ or otherwise ‘responsive to the receipt of the modem processor to application processor data,’ as required by claim 1.”). And, this understanding of “triggered by” is also supported by various dictionaries. *See* Ex-1412 [*Heritage Dictionary*], 1444 (“trigger[:] . . . 1. to set off; initiate”); Ex-1413 [*Oxford Dictionary*], 1541 (“trigger . . . cause to happen or exist”); Ex-1414 [*Merriam-Webster’s Dictionary*], 1337 (“trigger[:] . . . to initiate, actuate, or set off by a trigger”); Lin-¶67.

## **VIII. OVERVIEW OF PRINCIPAL PRIOR ART REFERENCES**

### **A. U.S. Patent No. 9,329,671 To Heinrich et al.**

U.S. Patent No. 9,329,671 to Heinrich et al. (Ex-1404) was filed January 29, 2013 and issued May 3, 2016. Heinrich is therefore prior art under at least 35 U.S.C. § 102(a)(2).

As shown below, Heinrich discloses a device with a baseband processor 104 and an application processor 106 that communicate with each other over an interconnectivity bus:



Ex-1404, Fig. 1, 4:26-46 (“FIG. 1 shows . . . a **baseband processor 104** . . . an **Application Processor (AP) 106** . . . [and] a physical interface configured for communicating IPC activities between the baseband processor 104 and the application processor 106.” (emphasis added)).

In Heinrich, data communications over the interconnectivity bus—referred to as “IPC [inter processor communication] activities”—are buffered so that transmission to the application processor can be delayed. *Id.*, 7:65-8:1 (“IPC activities that are not real-time sensitive . . . can be delayed until it is deemed

profitable to run them.”); *id.*, 8:14-15 (“Those IPC activities which are not real-time sensitive can be delayed.”). A scheduler implemented as software in the baseband processor includes “timers” that, upon expiration, trigger transmission of the buffered data over the bus to the application processor. *Id.*, 7:7-9 (“[T]here is a ***centralized scheduler 120*** which is associated with the baseband processor 104 . . . .” (emphasis added)); *id.*, 9:2-5 (“[T]he scheduler 120 allocates a respective ***timer***, herein referred to as a ‘***lazy timer***’, to each of the non real-time sensitive IPC activities identified in step S302. . . .” (emphasis added)); *id.*, 9:14-16 (“***When one of the lazy timers fires this causes the respective IPC activity to be communicated on the IPC interface to the application processor 106. . . .***” (emphasis added)); Lin-¶54.

Each data communication (“IPC activity”) is assigned a timer. Ex-1404, 9:2-6. When any one of the assigned timers expires (“fires”), the baseband processor sends the buffered data over the bus to the application processor. *Id.*, 9:11-21. Heinrich explains that, by grouping together the transmissions, the processor can reduce the number of times that the application processor enters and exits sleep mode, thereby reducing the power consumption of the computer system. *Id.*, 4:6-12; Lin-¶55.

Heinrich also discloses that the same technique—buffering data and sending it all at once upon expiration of a timer—can be used to control transmissions from

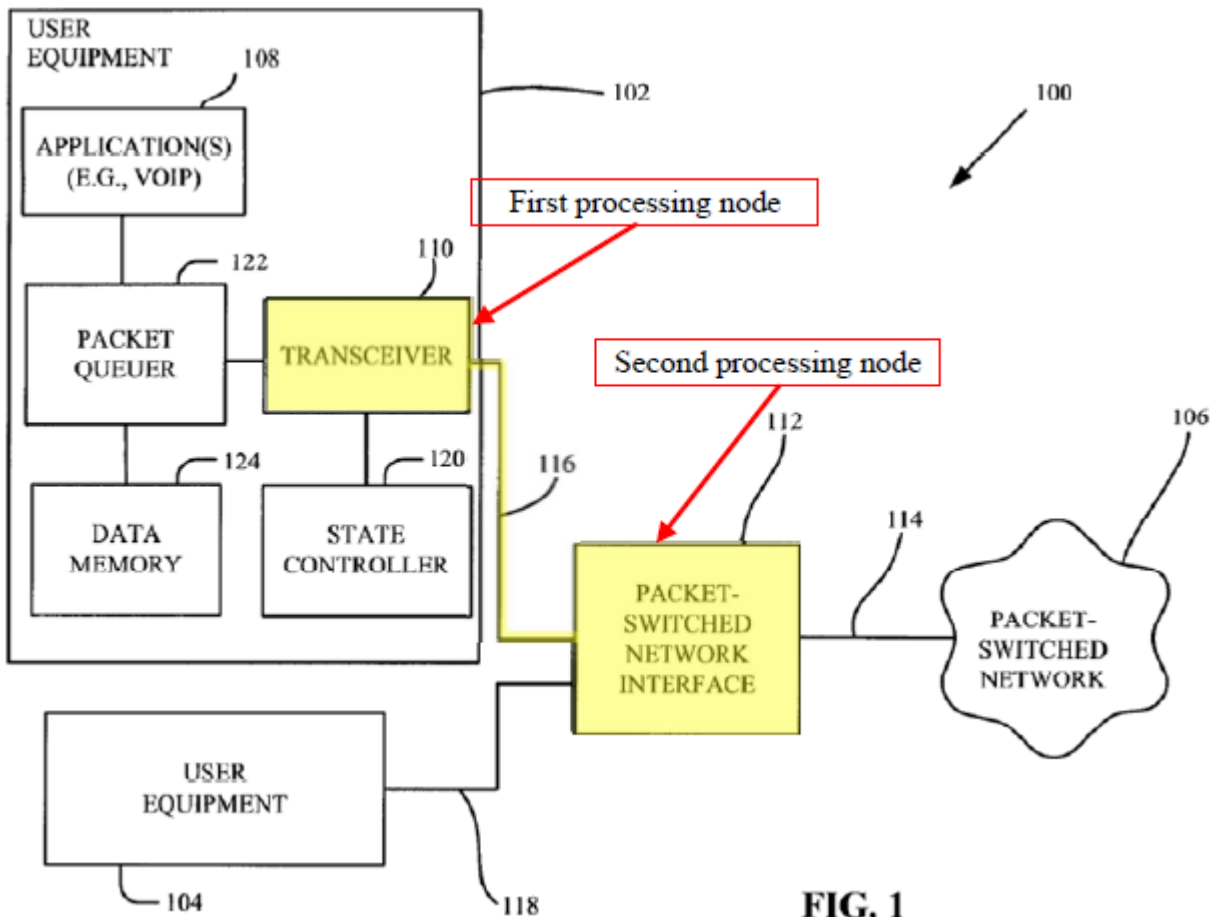
the application processor over the bus to the baseband processor. Ex-1404, 12:52-55 (“A *scheduler may implement the same scheduling techniques* as those described above, but configured to schedule IPC activities *from the application processor 106 to the baseband processor 104.*” (emphasis added)); Lin-¶56.

Heinrich also discloses that the scheduler of its baseband processor can detect that the physical IPC interface is in an active state, and the application processor is awake, when the baseband processor transmits data to the application processor. Ex-1404, 9:43-46 (“In a second example, the scheduler 120 can deduce that the application processor 106 will be in the awake mode by determining that real-time sensitive IPC activities are being sent to the application processor 106 . . . .”); *id.*, 9:54-62 (“The scheduler 120 can perform the determination that the application processor 106 is in the awake mode by receiving a notification that the physical IPC interface is in an active state. . . . [N]on real-time sensitive IPC activities are sent to the application processor 106 at a time when the application processor 106 is in the awake mode due to the communication of a previous real-time sensitive IPC activity.”); Lin-¶57.

**B. U.S. Patent No. 8,160,000 to Balasubramanian**

U.S. Patent No. 8,160,000 to Balasubramanian (Ex-1405) was filed October 3, 2006 and issued April 17, 2012. Balasubramanian is therefore prior art under at least 35 U.S.C. § 102(a)(1).

Balasubramanian describes a system where two processing nodes—for example, a transceiver 110 in a user device such as a cell phone, and a network interface 112 to a packet-switched network—communicate with each other over a communication link 116, which can be a wired connection:



Ex-1405, Fig. 1; Lin-¶59.

Balasubramanian discloses a scheme to synchronize transfers over the communication link 116, in which the transceiver 110 sends packets to the network interface 112, which *triggers* the network interface 112 to send any buffered

packets back to the transceiver 110 during the same active state of the transceiver 110. Ex-1405, 6:63-7:8 (“[T]he transceiver 110 ***then transmits the queued uplink packets over the communication link 116***. Advantageously, the queued packets may be grouped for transmission such that ***all of the packets are transmitted*** during a single wake state of the transceiver 110. . . . As represented by block 210, during the ***same single wake state the transceiver 110 also receives any downlink packets queued in the network interface 112***. For example, the network interface 112 may use the ***receipt of an uplink packet as a trigger*** to transmit any downlink packets in its queue.” (emphasis added)); *see also id.*, 2:23-24; Lin-¶60.

Balasubramanian discloses two processing nodes communicating over a wired link, like the bus in Heinrich. Ex-1405, 4:50-53 (“[T]he sub-network [which includes the transceiver 110 and the network interface 112] may communicate via some other protocol (*e.g.* a wire-based protocol or a wireless-based protocol) over communication links 116 and 118.”). Lin-¶61.

Like Heinrich and the ’490 patent, Balasubramanian is directed to conserving power by reducing the number of transitions from a low power mode to an active mode. Balasubramanian explains that a “transceiver” must transition from a “suspend” (low power) state to an “active” state to allow the transceiver to transmit its buffered data to the network interface. According to Balasubramanian, reducing the number of times the transceiver transitions from a suspend state into

an active state reduces power consumption. Ex-1405, 5:55-61 (“Here, ***power may be conserved*** by not transitioning the transceiver 110 from the suspended state to the active state every time a packet has been generated for transmission. . . . Accordingly, power savings may be achieved by rescheduling the packet traffic into groups of traffic.” (emphasis added)); Lin-¶62.

In order to reduce the number of power state transitions, Balasubramanian uses the same technique as the ’490 patent—buffering data intended for another processor during periods when the communication bus or other processor is inactive, and then later transmitting the buffered data from both processors during the same active state. Ex-1405, 5:66-6:4 (“[T]he user equipment 102 may include a packet queuer component 122 that facilitates queuing and temporarily ***storing the packets*** . . . . When the transceiver 110 is transitioned to an active state, the queued packets may be provided to the transceiver 110 for transmission to the network 106 (via interface 112).” (emphasis added)); *id.*, 6:65-67 (“Advantageously, the queued packets may be grouped for transmission such that ***all of the packets are transmitted during a single wake state*** of the transceiver 110.” (emphasis added)); Lin-¶63.

In addition to the “trigger” mechanism described above, Balasubramanian also uses a timer to determine when to transmit packets from the transceiver 110 to the network interface 112 (similar to Heinrich and the ’490 patent). Ex-1405,

9:33-34 (describing a “**timer**/counter 426 to determine when to make the packets in the packet queue 422 available. . . .” (emphasis added)); *id.*, 6:48-49 (“[T]he packets may be queued for configurable amount of time.”); *id.*, 6:55-60 (“[O]nce the **configurable amount of time has elapsed** or the configurable number of packets have been queued, the **transceiver 110 transitions** to an active (e.g., wake) state. . . . and **establish[es] communications with the network interface 112.**” (emphasis added)); *id.*, 9:7-9 (“[P]ackets may be queued for a configurable amount [of] time . . . .”); Lin-¶64.

## **IX. SPECIFIC GROUNDS FOR PETITION**

Pursuant to Rule 42.104(b)(4)-(5), the following sections describe in detail how the prior art discloses each and every limitation of the challenged claims of the ’490 patent, and how the prior art renders these claims obvious. *See also* Lin-¶¶71-137.

### **A. Ground 1: Claims 9, 11-13, 26, and 27 Are Rendered Obvious By Heinrich In View Of Balasubramanian**

Claims 9, 11, 12, and 13 depend from claim 1. While claim 1 is not challenged in this petition, a mapping of claim 1 is provided below because of these dependencies.

**1. Claim 1**

**a) [1a] Preamble: “[a] mobile terminal”**

The ’490 patent describes the claimed “mobile terminal” as a smart phone, cell phone, tablet, or similar device. Ex-1401, 6:37-41 (“The mobile terminal 22 may be a smart phone . . . , a cellular telephone, a tablet, a laptop, or other mobile computing device.”); Lin-¶72.

Heinrich discloses a “user device 102” that can be “a mobile phone, a tablet, a laptop computer or other embedded device able to connect to the network 110.” Ex-1404, 4:21-23; *see also id.*, Fig. 1 (showing user device 102); Lin-¶73.

**b) [1b] “a modem timer”**

Claim 1 requires “a modem timer.” Heinrich teaches a “scheduler” and, within the scheduler, a “timer.” The scheduler can be “associated with the baseband processor”—*i.e.*, the claimed “modem processor”—and controls communications between the baseband processor and the application processor. Ex-1404, 7:8-21 (“[T]here is a centralized scheduler 120 which is associated with the baseband processor 104 . . . . The scheduler 120 may control the scheduling of IPC activities [*i.e.*, processor-to-processor communications] in both directions between the processors 104 and 106 . . . .”); Lin-¶74.

The scheduler associated with the baseband processor includes a timer called a “lazy timer.” Ex-1404, 9:2-6 (“[T]he scheduler 120 allocates a respective *timer*,

herein referred to as a ‘*lazy timer*’, to each of the non real-time sensitive IPC activities identified in step S302. Each IPC activity is sent on the IPC interface when its respective lazy timer fires.” (emphasis added)). The “lazy timer” of Heinrich is a “modem timer” because it determines when data held by the baseband processor is sent to the application processor. *Id.*; *see also id.*, Fig. 1, 7:10-11 (“In the example shown in FIG. 1, the scheduler 120 is implemented as a software module on the baseband processor 104.”); Lin-¶75.

**c) [1c] “a modem processor”**

Heinrich discloses the “modem processor” of claim 1. For example, as shown in Figure 1, Heinrich discloses a mobile communication system that includes baseband processor 104:

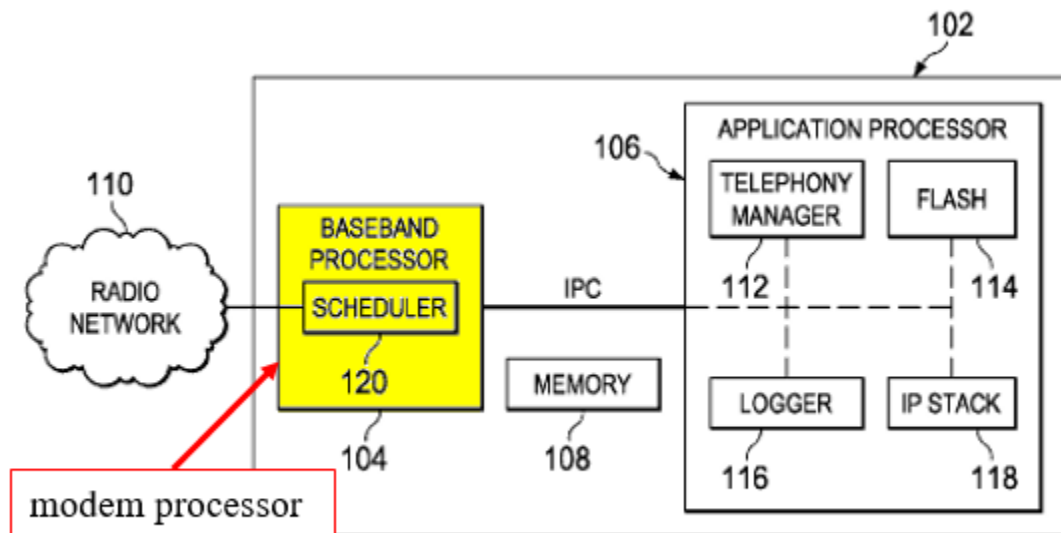


FIG. 1

Ex-1404, Fig. 1. Heinrich discloses that baseband processor 104 is a “modem” processor that communicates with a network. *Id.*, 4:30-36 (“The ***baseband processor 104 acts as a Radio Frequency (RF) modem*** to process data for communication between the user device 102 and the network 110 . . . [T]he baseband processor 104 is implemented at the user device 102 for communicating with the radio network 110.”). The modem processor of Heinrich—i.e., baseband processor 104—communicates with and receives data from a remote network (e.g., network 110). *Id.*; Lin-¶76.

**d) [1d] “the modem processor configured to hold modem processor to application processor data until expiration of the modem timer”**

Heinrich teaches that baseband processor 104 delays sending certain data to the application processor 106 over the IPC interface, and transmits such data upon expiration of a modem timer. *Id.*, 7:65-8:1 (“[T]he scheduler 120 identifies IPC activities that . . . can be ***delayed*** until it is deemed profitable to run them.”). These transactions are aggregated for later transmission. *Id.*, 9:5-13 (“Each IPC activity is sent on the IPC interface when its respective lazy timer fires . . . . However when one of the registered timers fires, ***all registered timers expire at the same time, causing all the aggregated IPC activities to be served at the same time.***” (emphasis added)). Heinrich also explains that buffering communications and sending them all at once when a timer expires results in “fewer transitions between

the sleep and awake modes” and thereby “reduce[s] the power consumed.” *Id.*, 10:44-51. One of ordinary skill in the art would understand that such delayed and “aggregated IPC activities” would be “modem processor to application processor data” that would have to be held (*i.e.*, buffered) until the baseband processor determines that it is time to transmit the held data to the application processor, such as when the timer expires. *See also id.*, 11:57-58 (“In order to delay file write accesses they are stored in a cache memory of the baseband subsystem.”); Lin-¶77.

Heinrich describes “baseband memory” where baseband data is buffered. Ex-1404, 11:58-66 (“The scheduler 120 schedules the file write accesses as described above such that a group of them may be retrieved from the cache memory of the baseband subsystem together and sent to the application processor 106 on the IPC interface as a group. . . These files may be cached in **baseband memory** with no user impact.” (emphasis added)); Lin-¶78.

It would have been obvious to a POSA that the baseband processor in Heinrich can hold its data—*i.e.*, the “modem processor to application processor data”—in its own memory on the same chip as the processor circuitry. A processor can hold data in two known ways, on-chip or off-chip. Both well-known ways of holding data are discussed, for example, in Panda et al., “On-Chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based

Systems,” *ACM Transactions on Design Automation of Electronic Systems*, 5(3):682-704 (July 2000) (“Panda”) (Ex-1411); Lin-¶79.

**First**, the processor could hold the data in its own memory—*i.e.*, on the same chip that includes the processor circuitry, such as in on-chip SRAM. Ex-1411, 683 (“The types of on-chip memory commonly integrated with the processor on the same chip are instruction cache, data cache, and on-chip SRAM . . . [I]t is possible to incorporate *embedded DRAMs* along with a processor core in the same chip . . . .”); Lin-¶80.

**Second**, the data could be held on a separate or external chip—such as a memory chip. *Id.*, 683 (“[A]ccess to off-chip memory (usually DRAM) requires relatively longer access times.”). *See also id.*, 686 (Fig. 2, showing offchip DRAM storage and onchip SRAM and data cache storage):

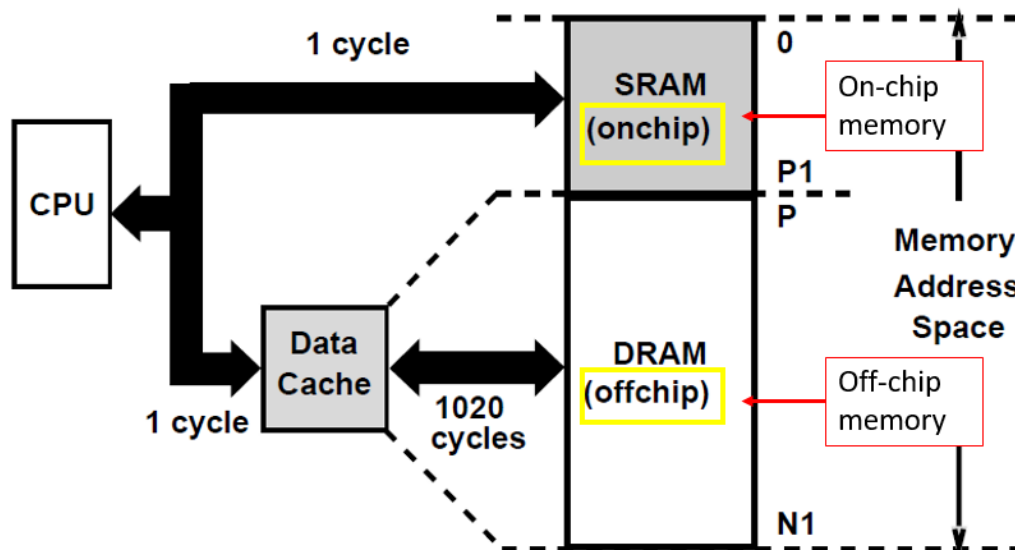


Fig. 2. Division of data address space between SRAM and DRAM.

A POSA would have been motivated to use “on-chip” memory to store data on the baseband processor in Heinrich for at least two reasons. First, using “on-chip” memory to store a processor’s data requires less physical space than using a separate chip dedicated to memory. *See id.*, 682-683. This allows for fewer chips in a device, reducing cost. *Id.*, 683. Second, accessing data from “on-chip” memory is much faster than accessing data from another chip over a separate connection, which improves device operation. *Id.* Because using on-chip memory to hold data was well-known in systems like Heinrich, a POSA would have reasonably expected the design to succeed. Lin-¶82. “When there is a design need or market pressure to solve a problem and there are a *finite number of identified, predictable solutions*, a POSA has good reason to pursue the known options within his or her technical grasp.” *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 421 (2007) (emphasis added).

**e) [1e] “an application processor”**

Heinrich discloses an “application processor.” As shown in Figure 1, Heinrich discloses a mobile communication system that includes an “application processor 106”:

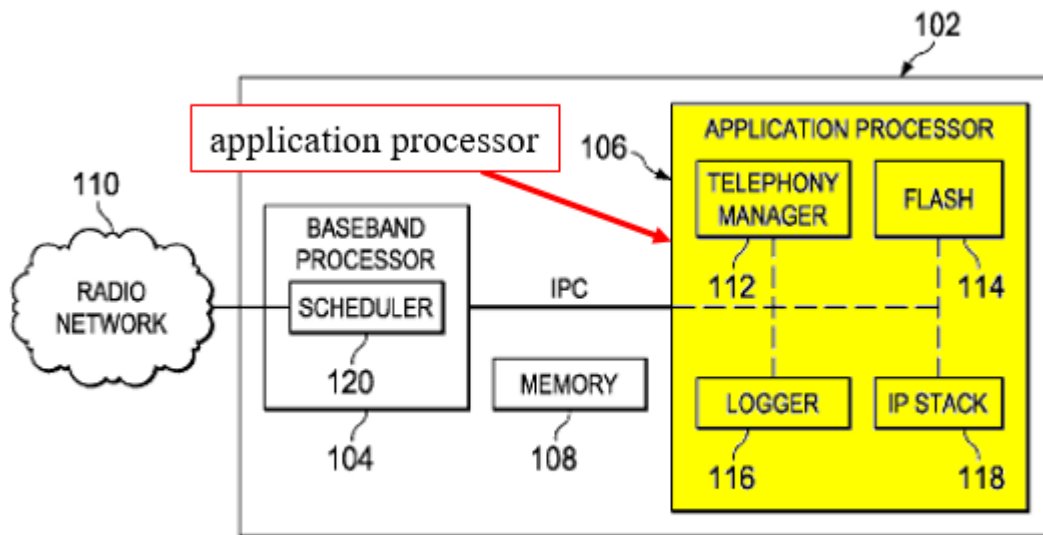


FIG. 1

Ex-1404, Fig. 1, 4:36-42 (“The *application processor 106* executes an operating system of the user device 102 and handles other multimedia features on the user device 102. For example, the application processor 106 processes data relating to peripherals (not shown in FIG. 1) of the user device 102 such as a display, a Wi-Fi module, a GPS module, etc.” (emphasis added)). Lin-¶83.

**f) [1f] “an interconnectivity bus communicatively coupling the application processor to the modem processor”**

Heinrich discloses a bus labeled “IPC” (“inter processor communication”) that communicatively couples the baseband processor 104—*i.e.*, modem processor—to the application processor 106:

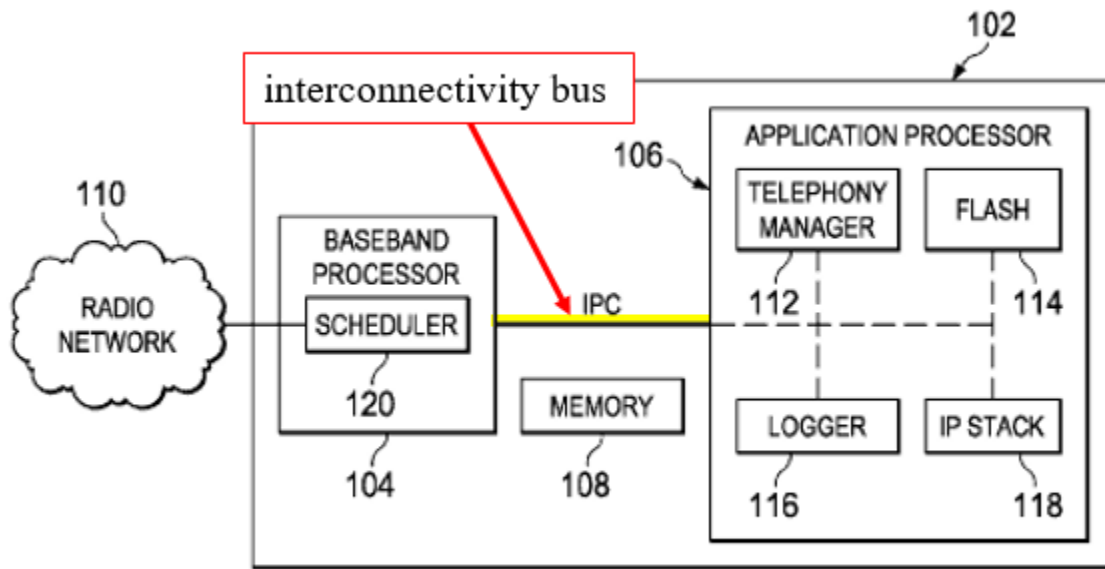


FIG. 1

Ex-1404, 4:44-46 (“There is a physical interface configured for communicating IPC activities between the baseband processor 104 and the application processor 106.”), Fig. 1; Lin-¶84.

The IPC bus in Heinrich is a “physical interface” that can be any one of many industry-standard buses such as “(i) a Universal Serial Bus (USB) interface, (ii) a Mobile Industry Processor Interface (MIPI), such as a High-Speed Synchronous Interface (HSI), (iii) a Serial Peripheral Interface (SPI), or (iv) a shared memory.” Ex-1404, 4:46-50; Lin-¶85.

- g) [1g] “the application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity**

**bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.”**

Claim [1g] requires “the application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.” Ex-1401, 17:63-18:5; Lin-¶86.

**“[T]he application processor configured to hold application processor to modem processor data”**: Heinrich renders obvious that the “application processor” is “configured to hold application processor to modem processor data.” As discussed above, Heinrich discloses that data is buffered prior to being sent over the bus from the baseband [modem] processor to the application processor. *See* claim [1d]. Heinrich further explains that the same methods used to transmit data from the baseband [modem] processor to the application processor—*i.e.*, buffering data and sending it all at once—can be used to transmit data from the application processor to the baseband [modem] processor. Ex-1404, 12:52-55 (“A scheduler may implement the *same scheduling techniques* as those described above [for scheduling IPC activities from the baseband processor 104 to the

application processor 106], but configured to schedule IPC activities *from the application processor 106 to the baseband processor 104.*” (emphasis added)); Lin-¶87.

It would have been obvious to a POSA to construct Heinrich’s application processor to hold the data for the same reasons described in claim [1d]. *See supra* claim [1d]. While Heinrich does not expressly specify where such data is held, it would have been obvious to a POSA that the application processor in Heinrich can hold its data—*i.e.*, the application processor to baseband processor data”—in its own memory on the same chip as the processor circuitry. Lin-¶88.

As explained above in claim limitation [1d], a POSA would have understood that there were two well-known ways for data to be buffered: (1) on the same chip that includes the processor circuitry; or (2) on a separate or external chip. *See* Ex-1411, 683; 686 (Fig. 2); Lin-¶89.

As with the baseband processor, a POSA would have been motivated to use “on-chip” memory to store data on the application processor in Heinrich for at least the two reasons identified above regarding claim limitation [1d].

**“[U]ntil triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus”:**

The combination of Balasubramanian and Heinrich discloses an application processor that holds data “until triggered by receipt of the modem processor to

application processor data from the modem processor through the interconnectivity bus.” Lin-¶91.

Heinrich teaches various techniques to determine when to send data from the application processor to the baseband processor. These techniques include (1) a timer that triggers transmission of data when the timer expires or (2) more generally, upon any other determination that informs the application processor that the baseband processor is in an “awake mode” and can therefore receive data. Ex-1404, 9:22-26 (“In general, each lazy timer is configured to fire in response to the earlier of: (i) the expiry of a respective deadline provided to the lazy timer before which it is expected to fire, or (ii) a determination that the application processor 106 is in the awake mode.”).<sup>4</sup>

Heinrich also discloses that the scheduler of its baseband processor can detect that the physical IPC interface is in an active state, and the application processor is awake, when the baseband processor transmits data to the application processor. *Id.*, 9:50-62 (“In order to determine that *real-time sensitive IPC*

---

<sup>4</sup> The quoted portion of Heinrich describes the transmission of data from the baseband processor. However, as explained above, Heinrich discloses that the same techniques can be used to control transmission of data from the application processor to the baseband processor. Ex-1404, 12:52-55.

*activities are being sent to the application processor 106*, the scheduler 120 registers to the underlying physical IPC interface in order to receive notifications of state changes of the IPC interface. The *scheduler 120 can perform the determination that the application processor 106 is in the awake mode by receiving a notification that the physical IPC interface is in an active state*. When the IPC interface enters an active state the scheduler 120 deems it appropriate to fire all registered lazy timers. In this way the non real-time sensitive IPC activities are sent to the application processor 106 at a time when the application processor 106 is in the awake mode due to the communication of a previous real-time sensitive IPC activity.” (emphasis added)). Because Heinrich notes that the scheduling techniques described with respect to the baseband processor are also applicable to the application processor (*see id.*, 12:52-55), Heinrich teaches that its application processor can include a scheduler that can detect that the physical IPC interface is in an active state, and the baseband processor is awake. *Id.*, 9:50-62, 12:47-64; Lin-¶93.

While Heinrich does not explicitly disclose holding data intended for transmission to another processor until receipt of data from the other processor, Balasubramanian discloses such a technique. In particular, Balasubramanian discloses holding (e.g., queuing), at a second processing node (e.g., network interface 112), data (e.g., downlink packets) to be sent to a first processing node

(e.g., transceiver 110) until “trigger[ed]” by “receipt” of data (e.g., uplink packets) from the first processing node over a bus (e.g., communication link 116):

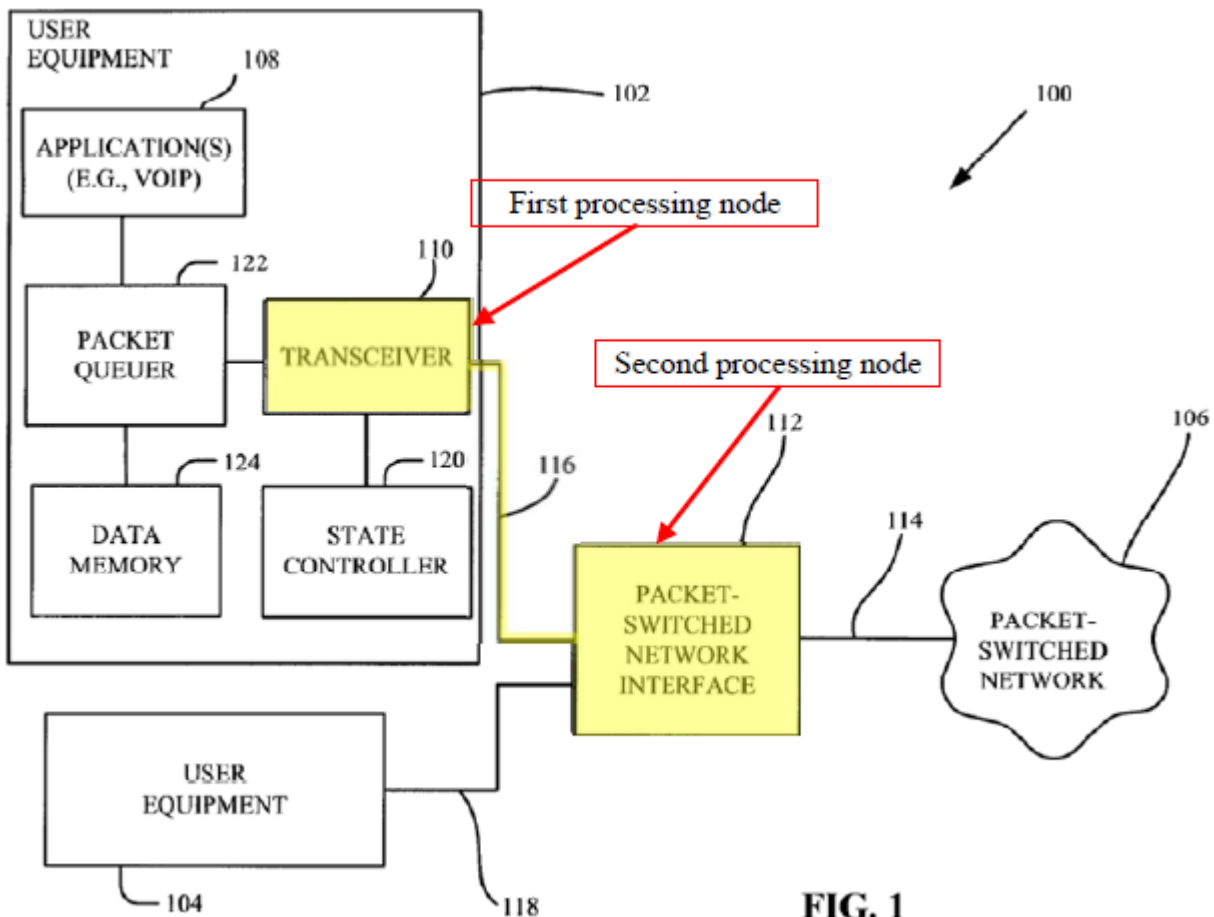


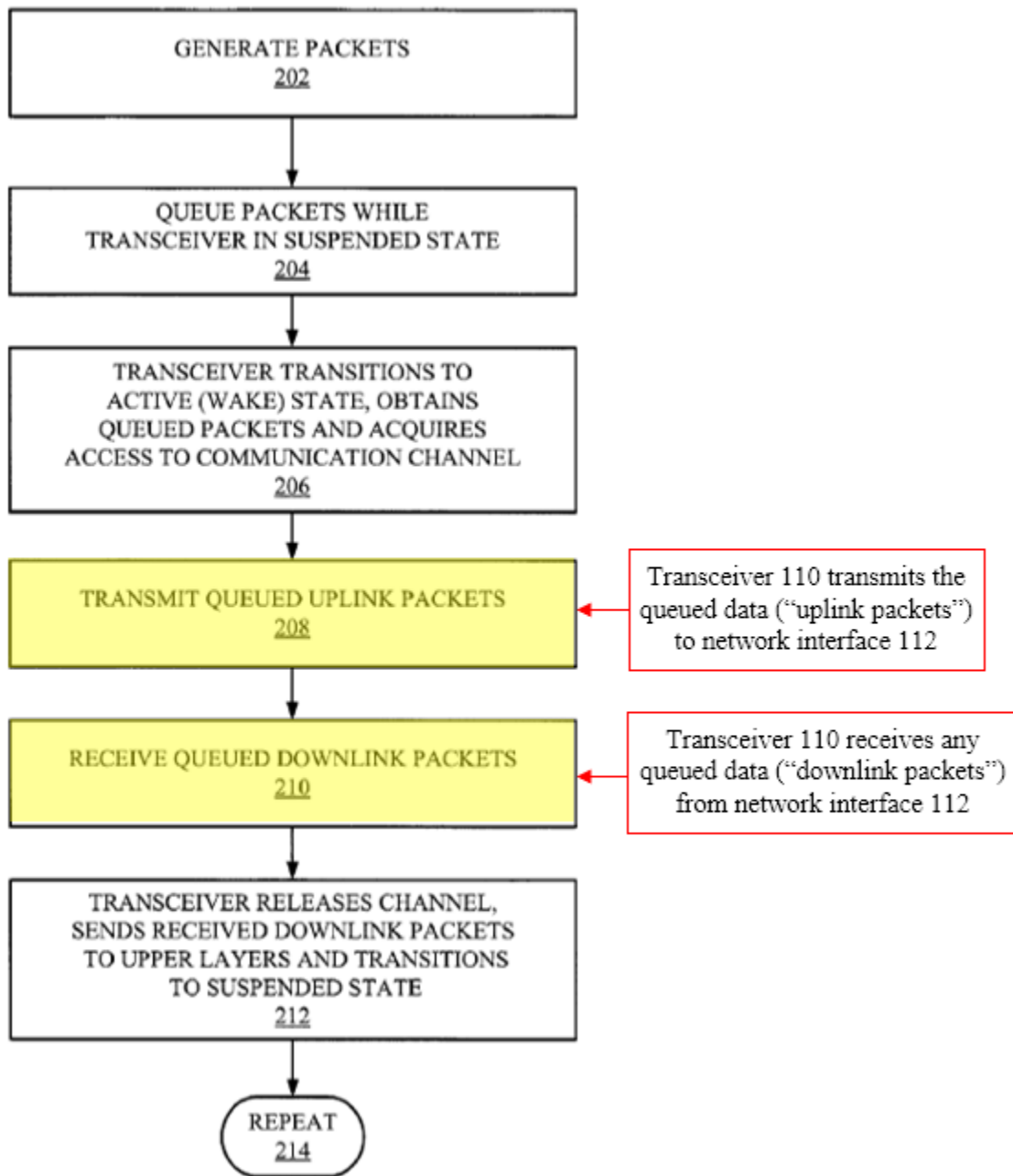
FIG. 1

Ex-1405, Fig. 1, 6:63-7:8 (“As represented by block 208, the *transceiver 110* then *transmits the queued uplink packets over the communication link 116*.”

Advantageously, the queued packets may be grouped for transmission such that *all of the packets are transmitted* during a single wake state of the transceiver 110. . . . As represented by block 210, during the same single wake state the

transceiver 110 also receives any downlink packets queued in the network interface 112. For example, the *network interface 112* may use the *receipt* of an *uplink* packet as a *trigger* to transmit any *downlink* packets in its queue.” (emphasis added)); Lin-¶94.

Figure 2 of Balasubramanian also discloses this feature. Consistent with the disclosure in Balasubramanian (*see* Ex-1405, 6:63-7:8), Figure 2 discloses that at step 208, the transceiver 110 transmits all its queued packets over the communication link 116 to the network interface 112. In response, as seen in step 210, the network interface 112 will transmit its queued packets to the transceiver 110.



**FIG. 2**

Ex-1405, Fig. 2, 6:63-7:8; Lin-¶95.

Accordingly, Balasubramanian discloses that the “trigger” for the transmission of data from the second processing node (network interface 112) to

the first processing node (transceiver 110) is the reception at the second processing node of all the queued packets from the first processing node. Lin-¶96. Thus, the combination of Heinrich and Balasubramanian teaches the claim limitation “until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.”<sup>5</sup> *Id.*

**“[A]fter which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus”:** The combination of Heinrich and Balasubramanian discloses that “after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.”

As explained above, Heinrich discloses a modem processor that transmits held data over an interconnectivity bus to the application processor upon expiration of a timer. Heinrich also discloses that its application processor can transmit held data to the modem processor upon determining that the modem processor is awake.

---

<sup>5</sup> This Petition sets forth reasons why a POSA would combine Heinrich and Balasubramanian in the section immediately below.

Ex-1404, 12:52-55 (“A scheduler may implement the *same scheduling techniques* as those described above, but configured to schedule IPC activities *from the application processor 106 to the baseband processor 104.*” (emphasis added)); 7:19-27 (“The scheduler 120 may control the scheduling of IPC activities in both directions between the processors 104 and 106. Alternatively . . . a *separate scheduler* (e.g. *implemented on the application processor*) controls the scheduling of IPC activities communicated *from the application processor 106 to the baseband processor 104.*” (emphasis added)); 9:50-57 (“In order to determine that *real-time sensitive IPC activities are being sent to the application processor 106,* the scheduler 120 registers to the underlying physical IPC interface in order to receive notifications of state changes of the IPC interface. The *scheduler 120 can perform the determination that the application processor 106 is in the awake mode by receiving a notification that the physical IPC interface is in an active state.*” (emphasis added)). Heinrich further discloses transmitting data from the application processor to the modem processor through a bus. *Id.*, 4:44-46. While each data communication (“IPC activity”) in Heinrich has its own timer, Heinrich discloses that all the held data in the modem processor is transmitted to the application processor upon expiration of any of the timers. *Id.*, 9:2-14 (“[T]he scheduler 120 allocates a respective timer, herein referred to as a ‘lazy timer’, to each of the non real-time sensitive IPC activities identified in step S302. . . .

However, when one of the registered timers fires, all registered timers expire at the same time, causing *all* the aggregated IPC activities to be served at the same time.” (emphasis added)); Lin-¶98.

Balasubramanian adds that receipt of data at a second processing node (network interface 112) from a first processing node (transceiver 110) “trigger[s]” the transmission of held data from the second processing node to the first processing node, meaning that the second processing node sends its data to the first processing node after receiving data from the first processing node and responsive to receiving data from the first processing node. Ex-1405, 7:4-8 (“As represented by block 210, during the same single wake state the transceiver 110 also receives any downlink packets queued in the network interface 112. For example, the *network interface 112* may use the *receipt* of an *uplink* packet as a *trigger* to transmit any *downlink* packets in its queue.” (emphasis added)); *id.*, 6:5-10 (“Similarly, the network interface 112 may be adapted to queue packets destined for the user equipment 102 when the transceiver 110 is in a suspended state. In this case, when the transceiver 110 is transitioned to an active state, the network interface 112 may send the queued packets to the transceiver 110.”); Lin-¶99.

Although Balasubramanian does not explicitly disclose a “modem processor” coupled to an “application processor” by an interconnectivity bus, it does disclose a first processing node (*e.g.*, transceiver 110) coupled to a second

processing node (*e.g.*, network interface 112) by an interconnectivity bus (communication link 116). *See* Ex-1405, Fig. 1. Balasubramanian further discloses transmitting data from a second processing node (*e.g.*, the network interface 112) to a first processing node (*e.g.*, transceiver 110) upon receipt of all the data from the first processing node. *Id.*, 6:63-7:8 (“As represented by block 208, the ***transceiver 110 then transmits the queued uplink packets over the communication link 116***. Advantageously, the queued packets may be grouped for transmission such that ***all of the packets are transmitted*** during a single wake state of the transceiver 110. . . . As represented by block 210, during the same single wake state the transceiver 110 also receives any downlink packets queued in the network interface 112. For example, the ***network interface 112*** may use the ***receipt*** of an ***uplink*** packet as a ***trigger*** to transmit any ***downlink*** packets in its queue.” (emphasis added)); Fig. 2; Lin-¶100. Finally, Balasubramanian teaches that its first processing node (*e.g.*, transceiver 110) can transmit its held data to the second processing node (*e.g.*, network interface 112) upon the expiration of a timer. Ex-1405, 6:55-60 (“As represented by block 206, once the ***configurable amount of time has elapsed*** or the configurable number of packets have been queued, the ***transceiver 110 transitions*** to an active (*e.g.*, wake) state. The transceiver 110 may thus obtain queued packets from the upper layers and

*establish communications with the network interface 112.*” (emphasis added));

Lin-¶100.

In other words, Heinrich teaches a modem processor that transmits all buffered data to an application processor upon expiration of a timer, and an application processor that transmits all buffered data to the modem processor upon determining that the modem processor is awake. Balasubramanian teaches a first processing node (*e.g.*, transceiver 110) that transmits data to a second processing node (*e.g.*, network interface 112) upon the expiration of a timer, and a second processing node that uses the receipt of the data from the first processing node as a trigger to send data back to the first processing node. Thus, Balasubramanian can be viewed as a particular application of the idea in Heinrich, because the trigger taught by Balasubramanian (the receipt of data from the first processing node) is just a specific way of determining that the first processing node is awake. Lin-¶101. Thus, the combination of Heinrich and Balasubramanian would result in a system in which a modem processor transmits all buffered data to an application processor upon expiration of a timer, and an application processor that transmits all buffered data to the modem processor upon determining, based on the receipt of data from the modem processor, that the modem processor is awake. *Id.* In other words, the combination would create a system in which the application processor is triggered to send its held data to the modem processor through an interconnectivity

bus, in response to the receipt of data from the modem processor through the interconnectivity bus. *Id.*

### **Obviousness**

**Combination of Heinrich and Balasubramanian:** It would have been obvious for a POSA to substitute the “trigger” scheme from Balasubramanian into the processor-to-processor communication system of Heinrich (which includes a “modem processor” and an “application processor”) in order to form the claimed combination. For example, as set forth above, Heinrich discloses the “modem processor,” “modem timer,” “application processor,” and “interconnectivity bus” limitations from claim 1. Balasubramanian teaches a very similar architecture as Heinrich—including a first processing node (transceiver 110), a timer (Ex-1405, 12:3-10), a second processing node (network interface 112), and a wired communication link connecting the first and second processing nodes (an “interconnectivity bus”). Moreover, Balasubramanian teaches the “trigger” scheme of claim 1, as set forth above. Accordingly, substituting the “trigger” scheme from Balasubramanian into the processor-to-processor communication system of Heinrich results in the claimed combination.

**Motivation to Combine:** A POSA would have been motivated to combine Balasubramanian’s “trigger” scheme with the processor-to-processor communication system of Heinrich for at least four reasons.

**First**, Heinrich and Balasubramanian are in the same field (communications between processing nodes) and are concerned with the same issues—power consumption when transitioning from low power-to-high power states. Ex-1404, 4:6-11 (“By grouping the non real-time sensitive IPC activities together and scheduling them for communicating to the second processor during a period in which the second processor is continuously in the first [active] mode, *the number of times that the second processor enters and exits the second mode (e.g. sleep mode) is reduced.*” (emphasis added)); Ex-1405, 5:55-61 (“Here, *power may be conserved* by not transitioning the transceiver 110 from the suspended state to the active state every time a packet has been generated for transmission. . . . Accordingly, power savings may be achieved by rescheduling the packet traffic into groups of traffic.” (emphasis added)). Further, as set forth above, both Heinrich and Balasubramanian teach similar architectures that include two processing nodes coupled by a bus, buffering data on both sides of the bus to allow for power savings states, and the use of a timer in one or both processing nodes to determine when to send queued data to the other processing node. Therefore, it would be natural for a POSA to look to Balasubramanian when considering the power consumption issues of Heinrich. Lin-¶104.

**Second**, Heinrich and Balasubramanian both solve the power consumption problem in the same way: minimizing the number of transitions from low power-

to-high power states. Thus, a POSA would have been motivated to use the techniques taught by Balasubramanian to address the same issues described in Heinrich. Lin-¶105.

**Third**, a POSA would realize that Heinrich is open to modification in that it discloses the detection of the state of the physical IPC interface in making decisions about when to transmit data. For instance, Heinrich notes that the scheduling techniques described with respect to its baseband processor are also applicable to the application processor (*see* Ex-1404, 12:52-55), and therefore Heinrich teaches that its application processor can include a scheduler that can detect that the physical IPC interface is in an active state, and the baseband processor is awake. *Id.*, 9:50-62, 12:47-64. Because Heinrich's application processor can have a scheduler that can detect when the physical IPC interface is in an active state, it would have been obvious to a POSA that the application processor in Heinrich could detect that the IPC interface and baseband processor are in an active state due to the reception of data from the baseband processor, upon which the application processor would transmit its data to the baseband processor over the IPC interface. Balasubramanian teaches just that—its network interface 112 detects that the transceiver 110 is in an active state upon receiving data from the transceiver 110, which triggers the network interface 112 to transmit its held data to the transceiver 110 over the communication link 116.

Balasubramanian also discloses that its teachings and techniques are not limited to a particular application. Ex-1405, 7:24-28. Lin-¶106.

**Fourth**, Heinrich recognizes that it is beneficial to transmit data from one processor—*e.g.*, an application processor—to a destination processor—*e.g.*, a baseband processor—when the destination processor is already awake/active. Ex-1404, 9:14-21 (“When one of the lazy timers fires this . . . wak[es] up the application processor 106. Therefore, this is a good time to schedule all the other pending, aggregated IPC activities to be sent to the application processor 106 . . . because . . . the application processor 106 is in the awake mode.”); 10:32-37 (“As described above, if the scheduler 120 can determine whether the application processor 106 is in a sleep mode, IPC activities may be scheduled without waking the application processor 106 from sleep mode by scheduling the IPC activities for times when the application processor 106 is already awake.”). The same principle applies to the trigger scheme of Balasubramanian, which teaches that the network interface 112 transmits packets to the transceiver 110 upon receiving packets from the transceiver 110—*i.e.*, when the transceiver 110 and communication link 116 are already awake. Therefore, consistent with the goals of Heinrich, a POSA would have been motivated to piggyback the transmission from the application processor back to the baseband processor during the same “active” mode time period and right after data is sent from the baseband processor to the application

processor, reducing the number of times the processor enters and exits the active mode. *See* Ex-1404, 4:6-11 (“By grouping the non real-time sensitive IPC activities together and scheduling them for communicating to the second processor during a period in which the second processor is continuously in the first [active] mode, the number of times that the second processor enters and exits the second mode (e.g. sleep mode) is reduced.”). Such a combination would have entailed merely incorporating a known technique (using the receipt of data from a processor as a way of determining that the other processor is awake) for a known purpose (to trigger a transmission to a processor upon receipt of data from the processor) with an expected result (fewer bus transitions between active mode and sleep mode).

Lin-¶107.

**Reasonable Expectation of Success:** A POSA would have had a reasonable expectation of success in combining Balasubramanian’s “trigger” scheme with the processor-to-processor communications system of Heinrich.

Heinrich discloses a system where an application processor can transmit held data to a baseband processor upon the occurrence of various triggering events. For example, Heinrich discloses a system where an application processor transmits

held data to a baseband processor upon expiration of a timer.<sup>6</sup> Ex-1404, 9:5-14 (“Each IPC activity is sent on the IPC interface when its respective lazy timer fires. . . . However when one of the registered timers fires, all registered timers expire at the same time, causing all aggregated IPC activities to be served at the same time.”). As another example, Heinrich discloses a system where an application processor transmits held data to a baseband processor when the application processor determines that the baseband processor is in an “active” or “awake” mode. *Id.*, 9:22-26 (“In general, each lazy timer is configured to fire in response to the earlier of: (i) the expiry of a respective deadline provided to the lazy timer before which it is expected to fire, or (ii) a **determination** that the

---

<sup>6</sup> While Heinrich focuses mostly on transmissions from the baseband processor to the application processor, Heinrich discloses that the same techniques can be used for transmissions from the application processor to the baseband processor. Ex-1404, 12:52-55 (“A scheduler may implement the same scheduling techniques as those described above, but configured to schedule IPC activities from the application processor 106 to the baseband processor 104.”). Thus, where Heinrich discloses a technique where a mobile processor can transmit held data to an application upon the occurrence of a trigger or event, it discloses that same feature for an application processor.

application processor 106 is in the awake mode.” (emphasis added)). Heinrich describes various techniques for determining that the other processor is awake, including: (1) receiving a notification that the bus has been placed in an awake mode (*id.*, 9:54-58); (2) receiving a notification via a hardware connection that the other processor is active (*id.*, 10:1-6); and (3) receiving a notification via a shared memory that the other processor is active (*id.*, 10:24-31). Thus, Heinrich discloses various techniques that trigger the transmission of data from one processor to another processor. Lin-¶109.

A POSA would expect that one could successfully use the receipt of data from one processing node to trigger a return transmission of data from another processor node, as disclosed in Balasubramanian. Viewed from the perspective of the disclosure in Heinrich, Balasubramanian’s technique is simply another way for one processor to determine that the other processor and bus are awake. A POSA would therefore understand that using Balasubramanian’s trigger scheme would work as expected in the processor-to-processor communications system of Heinrich. *Id.*, ¶110. In fact, Heinrich discloses that the “scheduler”—which determines when to transmit previously stored data from one processor to another—can detect when the baseband processor is active and trigger transmission of data from the application processor (or vice versa). Ex-1404, 10:1-5 (“In another example, the scheduler 120 can be notified when the remote

application processor 106 is active. In this way the scheduler 120 performs the **determination** that the application processor 106 is in the awake mode by **receiving a direct notification** of such.” (emphasis added)); *see also id.*, 9:43-49 (“[T]he scheduler 120 can deduce that the application processor 106 will be in the awake mode by determining that real-time sensitive IPC activities are being sent to the application processor 106, and on that basis can schedule the non real-time sensitive IPC activities to be communicated to the application processor 106 to make use of the awake mode of the application processor 106.”). Heinrich discloses that this notification “can be achieved through various means.” *Id.*, 10:5-6. A POSA would recognize that one such “means” is the trigger scheme described in Balasubramanian. Lin-¶110.

## 2. Claim 9

Claim 9 depends from claim 1, which as described *supra* in section IX.A is obvious over Heinrich in view of Balasubramanian. Heinrich renders obvious the limitation of claim 9, which recites that “the application processor comprises the modem timer.” *Id.*, ¶111.

As described in Section IX.A.1.b), the “lazy timer” of Heinrich is a “modem timer” because it determines when data held by the baseband processor is sent to the application processor. Ex-1404, 9:2-6; *see also id.*, Fig. 1, 7:10-11. Heinrich further teaches that the disclosed “lazy timers” may be allocated by a “scheduler.”

*Id.*, 9:2-6 (“[T]he scheduler 120 allocates a respective timer, herein referred to as a ‘lazy timer’, to each of the non real-time sensitive IPC activities identified in step S302.”); Lin-¶112.

In addition, Heinrich teaches that a single “scheduler” can control the scheduling of data transmissions in both directions between the baseband and application processors. *Id.*, 7:19-21 (“The scheduler 120 may control the scheduling of IPC activities in both directions between the processors 104 and 106.”); *id.*, 12:59-64. And, Heinrich teaches that a scheduler can be implemented on the baseband processor 104 or the application processor 106. *Id.*, 7:21-27 (“[A] **separate scheduler (e.g. implemented on the application processor)** controls the scheduling of IPC activities communicated from the application processor 106 to the baseband processor 104.”) (emphasis added); Lin-¶113.

While Heinrich does not explicitly disclose a scheduler on an application processor that controls the scheduling of data transmissions in both directions between processors 104 and 106, such an arrangement would have been obvious to a POSA in light of the above disclosures of Heinrich. As explained above, Heinrich teaches that a scheduler could be implemented on an application processor, and that a single scheduler can control the scheduling of data transmissions in both directions. It would have been obvious to a POSA to combine both concepts together in order to implement a scheduler on an

application processor that controls the scheduling of data transmissions in both directions. Such a scheduler would allocate a “lazy timer” that controls when data is sent from the baseband processor to the application processor (*i.e.*, a modem timer). Lin-¶114.

A POSA would have been motivated to use a single scheduler rather than two schedulers so as to simplify the design of the mobile device, and a POSA would have been motivated to place that scheduler on the application processor rather than the baseband processor, because there are only a handful of available placement options (on the baseband processor, on the application processor, or as a standalone chip), and the application processor has ample processing power to devote to such a task. A POSA would have had a reasonable expectation of success to modify Heinrich in this way, given that the combination is a combination of two simple concepts already taught in Heinrich. *Id.*, ¶115.

### **3. Claim 12**

Claim 12 depends from claim 1, which as described *supra* in section IX.A is obvious over Heinrich in view of Balasubramanian. Heinrich in view of Balasubramanian renders obvious the limitation of claim 12, which recites “a packet number limit counter associated with the modem processor, the modem processor configured to send data to the application processor if a threshold associated with the packet number limit counter is exceeded.” *Id.*, ¶116.

As described *supra* in Sections IX.A.1.d) and IX.A.1.g), Heinrich teaches a baseband processor sending data to an application processor upon the expiration of a lazy timer associated with the baseband processor. Heinrich arguably does not disclose sending data to the application processor “if a threshold associated with the packet number limit counter is exceeded.” Balasubramanian discloses “a packet number limit counter” and sending data “if a threshold associated with the packet number limit counter is exceeded.” Lin-¶117.

Balasubramanian discloses a “counter” that tracks a number of packets to queue before sending the packets. Ex-1405, 2:5-8 (“[P]ackets may be queued until the designated number of packets have been queued. At this point, the apparatus may transition to a wake state so that the queued packets may be transmitted as a group.”; *id.*, 6:55-65 (“[O]nce the configurable amount ... of packets have been queued, the transceiver 110 transitions to an active (e.g., wake) state. . . . [T]he transceiver 110 then transmits the queued uplink packets over the communication link 116.”); *id.*, 9:8-9 (“[A]configurable number of packets may be queued at a time.”); Lin-¶118.

It would have been obvious to modify Heinrich’s system in which data is sent at the expiration of a timer with Balasubramanian’s teaching of sending data “if a threshold associated with the packet number limit counter is exceeded.” Heinrich recognizes that there is a risk of buffer overflow if the buffer does not

flush its data prior to expiration of the lazy timer. For example, Heinrich explains that a logging buffer size should be larger than an amount of data accumulated before the expiration of a timer. Ex. 1404, 14:40-43 (“Instead of sending logging data out, the BB logger accumulates data in memory and starts an infinite lazy timer. *The logging buffer is big enough to accommodate more than 2 minutes of logging in idle mode.*” (emphasis added)). In addition to the reasons provided above for claim [1g], a POSA would have been motivated and had a reasonable expectation of success to modify Heinrich to send data based on reaching a packet count limit instead of or in addition to sending data based on the expiration of a timer given Heinrich’s recognition of the overflow risk. *See* Section IX.A.1.g). Further, including a packet counter into Heinrich’s system would have provided an alternative or additional guard against overfilling the buffer, would have allowed the use of a smaller buffer, and would have been obvious to try given the limited number of ways to track an amount of data that is buffered—*i.e.*, by time or size of the data. Lin-¶119. A POSA would have had a reasonable expectation of success from this combination because flushing a data buffer before it fills can be accomplished by sending data based either on a timer or buffer fill level. *Id.*

#### 4. Claim 11

Claim 11 depends from claim 1, which as described *supra* in section IX.A is obvious over Heinrich in view of Balasubramanian. Heinrich in view of

Balasubramanian renders obvious the limitation of claim 11, which recites “a byte accumulation limit counter associated with the modem processor, the modem processor configured to send data to the application processor if a threshold associated with the byte accumulation limit counter is exceeded.” Lin-¶120.

As described above with respect to claim 12, Heinrich in view of Balasubramanian renders obvious the additional limitation of a modem processor that sends data to an application processor if a threshold associated with the packet number limit counter is exceeded. For the same reasons set forth *supra* in Section IX.A.3, a POSA would also have found it obvious for the modem processor to send data to the application processor if a threshold associated with a byte accumulation limit counter is exceeded. Including a byte counter into Heinrich’s system, rather than a packet counter, would have provided an alternative or additional guard against overfilling the buffer in a system supporting variable packet sizes. In such systems, tracking a number of packets would not provide meaningful insight into the risk of buffer overflow, because a specific number of variable sized packets can occupy a wide range of memory sizes. Accordingly, a POSA would have been motivated in such systems to track the buffer fill level by monitoring the number of bytes rather than the number of packets. Lin-¶122. Further, expressing a buffer size in bytes rather than packets would have been obvious to try as there were only a finite number of ways to express the size of data (e.g., bits, bytes, or group of bits

and bytes) and Heinrich expressly recognized that data could be expressed in megabytes. *See* Ex-1404, 13:55-58 (“The following data were observed on a customer mobile phone design for a scenario in which, over a 36 hour period there is 4 hours of talk time, **500 MB** of download data and 100 texts, with the rest of the time being idle” (emphasis added)); Lin-¶122. A POSA would have had a reasonable expectation of success from this combination because flushing a data buffer before it fills can be accomplished by sending data based either on a timer or buffer fill level. *Id.*

## 5. Claim 13

Claim 13 depends from claim 1, which as described *supra* in section IX.A is obvious over Heinrich in view of Balasubramanian. Heinrich renders obvious the limitation of claim 13, which recites that the “modem processor is configured to determine if held data comprises a control packet and send such control packet before expiration of the modem timer.” *Id.*, ¶123.

Heinrich discloses that a “modem processor is configured to determine if ... data comprises a control packet and send such control packet before expiration of the modem timer.” As described above in claim [1d], Heinrich teaches accumulating certain data until the expiration of a “lazy timer.” *See* Section IX.A.1.d). More specifically, Heinrich discloses that “non real-time sensitive activities” are accumulated until expiration of a lazy timer, whereas real-time

sensitive activities are urgent and should not be delayed. Ex-1404, 8:14-20.

Heinrich also teaches a scheduler in the baseband processor determining whether IPC activities are real time sensitive and sending the real time sensitive activities prior the expiration of a lazy timer. *Id.*, 9:33-40 (“The procrastination property of the scheduler 120 as described in the example given above helps to synchronize non real-time sensitive IPC activities, however it does not synchronize these IPC activities with other IPC activities that are real-time sensitive, i.e. that can't wait before being scheduled. In step S302 *some of the IPC activities are determined as being real-time sensitive, and those IPC activities are communicated to the application processor 106 without delay.*” (emphasis added)); *id.*, FIG. 1.

Heinrich further teaches that IPC activities include “control information,” which Heinrich describes as activities associated with “initiating voice calls between the user device ... and another node of the radio network.” *Id.*, 5:4-6 . Heinrich also refers to “IPC activities” as “packets.” *See id.*, 13:61-62 (“Time to set IPC (USB) interface to sleep mode after last IPC packet: 1 s”). A POSA would have understood that “control information” would be a form of “real-time sensitive” data as voice data is usually processed without delay. Lin-¶124. A POSA would have also understood, given Heinrich’s teaching that IPC activities can be in the form of packets, that control information could take the form of one or more control packets. Taken together, Heinrich teaches the baseband processor

determining whether data is a real-time control packet and sending the real-time control packet to the application processor prior to the expiration of a lazy timer.

Heinrich arguably does not explicitly disclose that the “control information” is “held.” A POSA would have understood that even for “real-time” data, the scheduler in the baseband processor in Heinrich would have to buffer the data at least for as long as it took for the application processor to wake up. *Id.*, ¶125. Heinrich teaches that the application processor must be in the awake mode to receive and process data from the baseband processor. Ex-1404, 1:55-2:5 (“For example, the Application Processor may operate in an *awake mode* in which it is *able to process IPC activities that it receives from the baseband processor*. The Application Processor may alternatively operate in a *sleep mode* in which it *does not process IPC activities*.... Every time a quantum of data is sent over the IPC from the baseband processor to the Application Processor, the Application Processor needs to be in, or enter into, a state which allows for that communication to happen. If the Application Processor is in the sleep mode when the IPC activity is initiated then it is “woken up”, i.e. switched to operate in an awake mode in order to process the IPC activity.” (emphasis added)). Heinrich also explains that an application processor can take time to complete a transition from a low power mode to an awake mode. Ex-1404, 14:11-17 (“Assuming a typical DRX7 (1.28 s paging cycle) let us initially consider that both the application processor (AP) 106

and the baseband processor (BB) 104 are in low power mode. *T=0: first paging activity; BB wakes up to decode paging block*; BB sends logging data out; AP wakes; both AP and BB are awake; current~200 mA *T=1 s: AP completes exit from low power*”). A POSA would have also understood that this temporary holding of data would have been consistent with Heinrich’s teaching of sending real-time sensitive data without delay because Heinrich was not concerned with describing the lower level details of data processing—which include the general principle that data must be at least temporarily held in order to be processed. Lin-¶125.

**6. Claim 26**

**a) [26a] Preamble: “[a] mobile terminal”**

For the same reasons discussed above for claim [1a], Heinrich discloses the “mobile terminal” of claim 26. *See supra* Sections IX.A.1.a); Lin-¶126.

**b) [26b] “a modem byte accumulation limit counter”**

For the same reasons discussed above for claim 11, which recites “a byte accumulation limit counter associated with the modem processor,” Heinrich renders obvious the “modem byte accumulation limit counter” of claim 26. *See supra* Sections IX.A.3, IX.A.1.b, IX.A.4; Lin-¶127.

**c) [26c] “a modem processor”**

For the same reasons discussed above for claim [1c], Heinrich discloses the “modem processor” of claim 26. *See supra* Sections IX.A.1.c); Lin-¶128.

**d) [26d] “the modem processor configured to hold modem processor to application processor data until a predefined threshold of bytes has been reached by the modem byte accumulation limit counter”**

For the same reasons discussed above for claims [1d] and 11, which recite “the modem processor configured to hold modem processor to application processor data until expiration of the modem timer” and “the modem processor configured to send data to the application processor if a threshold associated with the byte accumulation limit counter is exceeded,” respectively, Heinrich renders obvious the “the modem processor configured to hold modem processor to application processor data until a predefined threshold of bytes has been reached by the modem byte accumulation limit counter” of claim 26. *See supra* Sections IX.A.1.d), IX.A.3, IX.A.1.b; Lin-¶129.

**e) [26e] “an application processor”**

For the same reasons discussed above for claim [1e], Heinrich discloses the “application processor” of claim 26. *See supra* Sections IX.A.1.e); Lin-¶130.

**f) [26f] “an interconnectivity bus communicatively coupling the application processor to the modem processor”**

For the same reasons discussed above for claim [1f], Heinrich discloses the “interconnectivity bus communicatively coupling the application processor to the modem processor” of claim 26. *See supra* Sections IX.A.1.f), IX.A.1.b; Lin-¶131.

**g) [26g] “the application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.”**

For the same reasons discussed above for claim [1g], Heinrich in view of Balasubramanian renders obvious the “application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus” of claim 26. *See supra* Sections IX.A.1.g), IX.A.1.b; Lin-¶132.

**7. Claim 27**

**a) [27a] Preamble: “[a] mobile terminal”**

For the same reasons discussed above for claim [1a], Heinrich discloses the “mobile terminal” of claim 27. *See supra* Sections IX.A.1.a); Lin-¶133.

**b) [27b] “a modem packet counter”**

For the same reasons discussed above for claim 12, which recites “packet number limit counter associated with the modem processor,” Heinrich in view of Balasubramanian renders obvious the “modem packet counter” of claim 27. *See supra* Sections IX.A.3, IX.A.1.b, IX.A.1.c; Lin-¶134.

**c) [27c] “a modem processor”**

For the same reasons discussed above for claim [1c], Heinrich discloses the “modem processor” of claim 27. *See supra* Sections IX.A.1.c), IX.A.1.b; Lin-¶135.

**d) [27d] “the modem processor configured to hold modem processor to application processor data until a predefined threshold of packets has been reached by the modem packet counter”**

For the same reasons discussed above for claims [1d] and 12, which recite “the modem processor configured to hold modem processor to application processor data until expiration of the modem timer” and “the modem processor configured to send data to the application processor if a threshold associated with the packet number limit counter is exceeded,” respectively, Heinrich in view of

Balasubramanian renders obvious the “the modem processor configured to hold modem processor to application processor data until a predefined threshold of packets has been reached by the modem packet counter” of claim 27. *See supra* Sections IX.A.1.d), IX.A.3, IX.A.1.b; Lin-¶136.

**e) [27e] “an application processor”**

For the same reasons discussed above for claim [1e], Heinrich discloses the “application processor” of claim 27. *See supra* Sections IX.A.1.e), IX.A.1.b; Lin-¶137.

**f) [27f] “an interconnectivity bus communicatively coupling the application processor to the modem processor”**

For the same reasons discussed above for claim [1f], Heinrich discloses the “interconnectivity bus communicatively coupling the application processor to the modem processor” of claim 27. *See supra* Section IX.A.1.f), IX.A.1.b; Lin-¶138.

**g) [27g] “the application processor configured to hold application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus.”**

For the same reasons discussed above for claim [1g], Heinrich in view of Balasubramanian renders obvious the “application processor configured to hold

application processor to modem processor data until triggered by receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus after which the application processor to modem processor data is sent to the modem processor through the interconnectivity bus responsive to the receipt of the modem processor to application processor data from the modem processor through the interconnectivity bus” of claim 27. *See supra* Section IX.A.1.g), IX.A.1.b; Lin-¶139.

## **X. CONCLUSION**

Based on the foregoing, claims 9, 11-13, 26, and 27 of the '490 patent recite subject matter that is obvious. The Petitioner requests institution of an *inter partes* review to cancel those claims.

Respectfully submitted,

Intel Corp.,  
Petitioner

By:       /Jason Kipnis/        
Jason D. Kipnis  
Registration No. 40,680  
Wilmer Cutler Pickering  
Hale and Dorr LLP

**CERTIFICATE OF SERVICE**

I hereby certify that on July 6, 2018, I caused a true and correct copy of the following materials:

- Petition for *Inter Partes Review* of U.S. Patent No. 9,535,490 Under 35 U.S.C. § 312 and 37 C.F.R. § 42.104
- Exhibits 1401-1417
- List of Exhibits for Petition for *Inter Partes* Review of U.S. Patent No. 9,535,490 (Exhibits 1401-1417)
- Power of Attorney
- Fee Authorization
- Certificate of Compliance Regarding Word Count

to be served via Federal Express on the following attorney of record as listed on PAIR:

W&T/Qualcomm  
106 Pinedale Springs Way  
Cary NC 27511

          /Jason Kipnis/          

Jason D. Kipnis  
Registration No. 40,680

**CERTIFICATE OF COMPLIANCE**

I hereby certify that the foregoing, Petition for *Inter Partes Review* of U.S. Patent No. 9,535,490, contains **13,379** words as measured by the word processing software used to prepare the document, in compliance with 37 C.F.R. § 42.24 (d).

Respectfully submitted,

Dated:

/Jason Kipnis/  
Jason D. Kipnis  
Registration No. 40,680

**LIST OF EXHIBITS FOR**  
**PETITION FOR *INTER PARTES* REVIEW OF**  
**U.S. PATENT NO. 9,535,490**

<b><u>Exhibit</u></b>	<b><u>Description</u></b>
1401	U.S. Patent No. 9,535,490
1402	Declaration of Dr. Bill Lin
1403	File History for U.S. Patent No. 9,535,490
1404	U.S. Patent No. 9,329,671 to Heinrich et al. (“Heinrich”)
1405	U.S. Patent No. 8,160,000 to Balasubramanian (“Balasubramanian”)
1406	PCT Publication No. WO 2009/039034 to Tsai (“the Intel PCT”)
1407	U.S. Patent No. 6,021,264 to Morita (“Morita”)
1408	Kaplan, Wiley Electrical and Electronics Engineering Dictionary, IEEE Press, John Wiley & Sons, 2004 (“Kaplan”)
1409	Downing, et al, Dictionary of Computer and Internet Terms, Barron’s Educational Series, Inc., 2013 (“Downing”)
1410	Duan et al., “Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic,” SRUTI. July 7, 2005 (“Duan”)
1411	Panda et al., “On-Chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based Systems,” ACM Transactions on Design Automation of Electronic Systems, Vol. 5, No. 3, July 2000, (“Panda”)
1412	Houghton Mifflin Co., The American Heritage College Dictionary, 2000 (“Heritage Dictionary”)
1413	Soanes et al., Concise Oxford English Dictionary, 11th Ed., Oxford University Press, 2004 (“Oxford Dictionary”)
1414	Merriam-Webster, Inc., Merriam-Webster's Collegiate Dictionary, 11th Ed., 2000 (“Merriam-Webster's Dictionary”)
1415	Gerber, et al., “P2P the gorilla in the cable,” National Cable & Telecommunications Association (NCTA) 2003 National Show, June 2000 (“Gerber”)

U.S. Patent No. 9,535,490  
Petition for *Inter Partes* Review

<b><u>Exhibit</u></b>	<b><u>Description</u></b>
1416	Kolbehdari, et al., “The Emergence of PCI Express in the Next Generation of Mobile Platforms,” Intel Technology Journal, Vol. 9, No. 1, 2005 (“Kolbehdari”)
1417	U.S. Patent No. 8,112,646 to Tsai (“Tsai”)