



- (51) **International Patent Classification:**
G06F 13/14 (2006.01) *G06F 13/36* (2006.01)
- (21) **International Application Number:**
PCT/US2013/063647
- (22) **International Filing Date:**
7 October 2013 (07.10.2013)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
13/648,139 9 October 2012 (09.10.2012) US
- (71) **Applicant (for all designated States except IR, SA, US):**
INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).
- (72) **Inventors; and**
- (71) **Applicants (for US only):** **BORKOWSKI, Dan G.** [US/US]; 414 Sunnyhill Road, Lunenburg, Massachusetts 01462 (US). **SISTLA, Krishnakanth V.** [IN/US]; 6000 SW Fountain Grove Terrace, Beaverton, Oregon 97007 (US).
- (74) **Agent:** **BOONE, Carrie A.**; Law Office of Carrie A. Boone, PC, C/O CPA Global, P.O. Box 52050, Minneapolis, Minnesota 55402 (US).

(81) **Designated States** (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

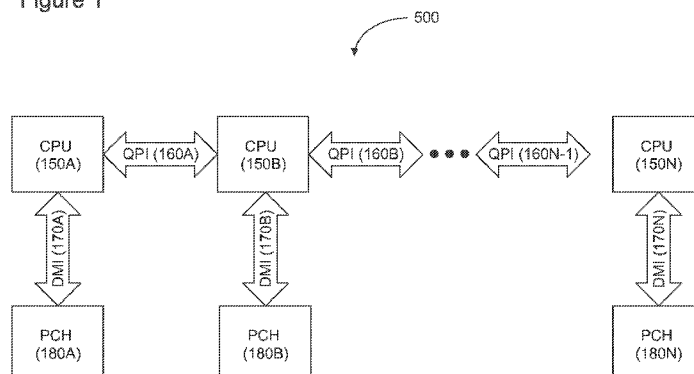
(84) **Designated States** (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— with international search report (Art. 21(3))

(54) **Title:** VIRTUALIZED COMMUNICATION SOCKETS FOR MULTI-FLOW ACCESS TO MESSAGE CHANNEL INFRASTRUCTURE WITHIN CPU

Figure 1



(57) **Abstract:** A message channel optimization method and system enables multi-flow access to the message channel infrastructure within a CPU of a processor-based system. A user (pcode) employs a virtual channel to submit message channel transactions, with the message channel driver processing the transaction "behind the scenes". The message channel driver thus allows the user to continue processing without having to block other transactions from being processed. Each transaction will be processed, either immediately or at some future time, by the message channel driver. The message channel optimization method and system are useful for tasks involving message channel transactions as well as non-message channel transactions.

**VIRTUALIZED COMMUNICATION SOCKETS FOR MULTI-FLOW ACCESS TO
MESSAGE CHANNEL INFRASTRUCTURE WITHIN CPU**

TECHNICAL FIELD

5 This application relates to multiprocessor systems, portable machine code, and message channel transaction processing.

BACKGROUND

Figure 1 is a simplified block diagram of a multiprocessor system 500, according to some embodiments. The multiprocessor system 500 includes N central processing units (CPUs) 150A, 150B, ..., 150N (collectively, "CPUs 150"), which are coupled to N – 1 specialized busses, known as quick path interconnect (QPI) busses 160A, 160B, ..., 160N-1 (collectively, "QPI busses 160"). The QPI busses 160, specifically designed for the CPUs, speed up communication between the CPUs 150. The CPUs may also be coupled to one or more volatile memories (not shown).

15 Also featured in the multiprocessor system 500 are up to N peripheral controller hubs (PCHs) 180A, ..., 180N (collectively, "PCHs 180") coupled to the CPUs 150 via up to N specialized busses, known as direct media interface (DMI) busses 170A, 170B,..., 170N. The PCHs 180 interface between the CPUs 150 and one or more peripheral devices of the multiprocessor system 500. The PCHs 180 may include display, input/output (I/O) control, a real-time clock, and other functions and may connect to an integrated display as well as other peripheral devices, such as a keyboard, a mouse, a non-volatile storage device, and so on.

For communication between endpoints within the processor of a multiprocessor system 500 or a single processor-based system, a message channel is used. The message channel is the transmission medium for these communications, and may be thought of as a type of "tunnel" or "subway" between endpoints inside the processor. There may be many message channel endpoints, and a message may be sent from any endpoint to any other endpoint, with the endpoints being functional entities within the processor. Portable machine code, or pcode, is used to communicate between the entities, and the pcode has its own endpoint for sending messages to other endpoints. (No endpoint sends an autonomous message to the pcode, as the only message that is received by the pcode endpoint is a response to a message that the pcode originated.) Power management request (PMReq) messages go to other entities using the QPI bus, which is similar to the message channel, except that the QPI bus is an external bus/interface. The message channel, by contrast, is strictly internal to the processor.

In CPU-based systems, such as a single-processor system or the multiprocessor system 500 of Figure 1, a message channel is used by many disparate pcode flows and functions. These functions may be used to read and write uncore control registers, issue PMReqs, and send messages to other platform entities (e.g., other CPUs 150, PCHs 180). The pcode uses the message channel quite frequently, from hundreds of times per millisecond to thousands of times per millisecond.

Some newer multiprocessor systems are designed in such a way that the message channel may become blocked at various times, such as during a frequency transition. Previous multiprocessor systems did not have this issue, as their message channel interfaces were always fully functional. So, the pcode in previous projects could use the message channel in a "blocking" manner by sending the transaction onto the message channel, and waiting in a tight loop for the completion of the transaction.

For newer multiprocessor systems, the use of "blocking" transactions on the message channel is deemed unacceptable because the blocking transaction can potentially lock up pcode for several tens of microseconds. The blocking transactions thus lead to a higher latency for other (non-message-channel-related) functions and impact the performance of the CPU. In addition, there is a risk of a deadlock because the message channel is blocked by some function that is waiting for something from the pcode via a sideband interface, but the pcode is blocked waiting for a message channel transaction to complete.

Additionally, PMreq messages require arbitration for use of a single buffer in a PMReq engine (PME). PMreq messages go over the message channel to the PME, and then over the QPI bus 160 to another CPU 150 (or over the DMI bus 170 to the PCH 180). As part of the PMreq protocol correctness, the PME will wait for a completion (CMP) from the other CPU/PCH, and will keep the PMReq buffer locked until the completion is actually received. In this case, if a blocking message channel transaction is used, the pcode will be locked up for the entire round-trip duration of the PMreq/CMP exchange. There may be delays on the other CPU (due to a frequency change, etc.), which further prolongs the duration of the lock-up.

Thus, there is a continuing need for a solution that overcomes the shortcomings of the prior art.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of this document will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein like reference numerals refer to like parts throughout the various views, unless otherwise specified.

Figure 1 is a simplified block diagram of a multiprocessor system, according to some embodiments;

Figure 2 is a simplified block diagram of the message channel optimization method and system, according to some embodiments;

5 Figure 3 is a flow diagram showing task flow for two tasks, according to some embodiments;

Figure 4 is a flow diagram showing task flow for the same two tasks as in Figure 3, this time using the method and system of Figure 2, according to some embodiments;

10 Figure 5 is a flow diagram following a message channel send operation as processed by the message channel optimization method and system of Figure 2, according to some embodiments; and

Figure 6 is a flow diagram following a message channel read operation as processed by the message channel optimization method and system of Figure 2, according to some embodiments.

DETAILED DESCRIPTION

15 In accordance with the embodiments described herein, a virtual message channel and driver are disclosed for enabling multi-flow access to the message channel infrastructure within a CPU of a processor-based system. Each pcode flow that uses the message channel is assigned a virtual channel identifier. Each flow with a virtual channel identifier can register a callback address and can submit one message channel transaction for processing. The transaction will be
20 processed, either immediately or at some future time, by a message channel driver.

In the following detailed description, reference is made to the accompanying drawings, which show by way of illustration specific embodiments in which the subject matter described herein may be practiced. However, it is to be understood that other embodiments will become apparent to those of ordinary skill in the art upon reading this disclosure. The following detailed
25 description is, therefore, not to be construed in a limiting sense, as the scope of the subject matter is defined by the claims.

Figure 2 is a simplified block diagram of a message channel optimization method 100, or optimization method 100, according to some embodiments. The optimization method 100 employs virtualized communication sockets for enabling multi-flow access to the message
30 channel infrastructure of a processor-based system. The optimization method 100 uses a message channel driver 50 to interface between virtual channels 70A, 70B, ..., 70K (collectively, "virtual channels 70") and a physical message channel 20 that is used for communication between endpoints within a processor of the processor-based system.

In some embodiments, the optimization method 100 solves the issues described in the background section for systems such as the multiprocessor system 500 of Figure 1, by using the virtual message channels 70 rather than blocking use of the message channel 50. Although the optimization method 100 addresses a problem found in multiprocessor systems, the method may also be used in single-processor systems.

Users, denoted as user 1 30A, user 2 30B, ..., user K 30K (collectively, "users 30"), represent portions of the pcode flow throughout the multiprocessor system. Tools 80, consisting of application programming interface (API) and helper functions, are available to both the message channel driver 50 and to the users 30.

In some embodiments, the method 100 assigns a virtual channel identifier (VCID) 60 to each user (pcode flow) 30 that uses the message channel 20. Virtual channel identifiers 60A, 60B, ..., 60K (collectively, "VCIDs 60") are available to each user 30A, 30B, ..., 30K, respectively. Each pcode flow 30 with a VCID 60 is also capable of registering a callback address. Callback addresses 40A, 40B, ..., 40K are featured in Figure 2 for users 1, 2, ..., K, respectively (collectively, "callback addresses 40"). The callback address 40 for each user 30 enables the message channel driver 50 to return to the pcode flow at the appropriate address, following processing of the message channel transaction for that user. Each user 30 with a VCID 60 can submit one message-channel transaction for processing.

In some embodiments, each user 30A, 30B, ..., 30K includes a respective buffer 90A, 90B, ..., 90K (collectively, "buffers 90"). The buffers 90 are temporary storage for message channel data that is to be returned following a message transaction. In some embodiments, the message channel driver 50 retrieves the returned data to the respective buffer 90 when processing message channel transactions.

Further, in some embodiments, the message channel driver 50 includes a queue 110 to keep track of transactions that are waiting to be processed. When processing a succeeding transaction, the message channel driver will retrieve the transaction from the queue 110. In some embodiments, the queue 110 is a first-in-first-out queue.

The users 30 of Figure 2 are presumed to be those pcode flows in which message channel transactions are to take place. Other pcode flows may not engage in message channel transactions. Such pcode flows are thus not assigned a VCID 60, in some embodiments. Nevertheless, as is shown below, in some embodiments, pcode flows for both message channel transactions and non-message channel transaction benefit from the optimization method 100 described herein.

In some embodiments, the message channel transaction is processed, either immediately or at some future time, by the message channel driver 50. The message channel driver 50 registers

an event that causes a kernel within the processor-based system to run a callback function when the message channel transaction has completed. The user (pcode flow) 30 that submitted the transaction becomes notified of the completion when its callback function runs, enabling the user to take any further desired actions subsequent to the completion.

Figure 3 is a flow diagram illustrating a method 200 in which two tasks are sequentially processed by the message channel 20, according to some embodiments. The operations 200 of Figure 3 are performed either in a single processor-based system or in a multiprocessor system, such as the system 500 of Figure 1. The method 200 features multiple steps being performed in time periods denoted to the left of each step in red. Two tasks, task 1 and task 2, are to be performed. The tasks are part of the pcode flow 30 referred to earlier, with the pcode flow issuing a message channel transaction in a single processor-based system or in the multiprocessor system 500. Task 1 involves a message channel transaction while task 2 does not.

For those tasks involving message channel transactions, the task is divided into 1) preparation of data, 2) a transaction on the message channel, 3) processing of the results of the message channel, and 4) other data processing that may not be message channel related.

At a first time period, (time period 1), task 1, a first pcode flow (user) 30, begins (block 200). Following data preparation (block 202) at time period 2, the pcode flow 30 issues a message channel transaction at time period 3 (block 204), thus invoking the message channel 20. The first pcode flow 30 waits for the transaction to complete (block 206), with no further processing taking place. In this example, there is a timeout period of 100 time units, and the pcode flow 30 is blocked during that entire period, known as a blocking transaction. At time period 103, the results of the message transaction are processed (block 208), and task 1 is completed (block 210).

The pcode flow 30 further includes a second task, task 2. Task 2 commences at time period 106 (block 212). Data processing takes place at time period 107 (block 214), and task 2 is completed by time period 108 (block 216), with additional tasks to be performed following the completion of task 2.

Figure 4, by contrast, features the same two tasks being sequentially performed on either a single processor-based system or on a multiprocessor system such as the system 500 of Figure 1, this time using the message channel optimization method 100 of Figure 2, according to some embodiments. The first task, task 1, however, is separated into two parts, task 1a and task 1b, which are processed separately.

Initially, the first part of the first task, task 1a, is processed in a manner similar to how task 1 was processed in Figure 3. At a first time period, (time period 1), task 1, a first pcode flow 30 begins (block 300). Following data preparation at time period 2 (block 302), the pcode flow 30

issues a message channel transaction at time period 3 (block 304), thus invoking the message channel 20. In this example, rather than having the first pcode flow 30 wait for the transaction to complete (as in block 206 of Figure 3), the message channel transaction is queued by the message channel driver 50 in time period 4 (block 306). Task 1a, the first part of task 1, is
5 completed at time period 5 (block 308).

As with the method 200 (Figure 3), in the method 300 (Figure 4), the pcode flow 30 further includes the second task, task 2. Task 2 commences at time period 6 (block 310). Data processing takes place at time period 7 (block 312), and task 2 is completed by time period 8 (block 314), with additional tasks to be performed following the completion of task 2. Note that,
10 in Figure 3, task2 completed at time period 108, while, in the current example, task 2 completed one hundred time units sooner, at time period 8.

Returning to the processing of the first task, when the message channel transaction is issued(block 304), the transaction is being processed, just as in Figure 3. And, just as in the Figure 3, the transaction uses 100 time periods to be processed. In Figure 4, the transaction
15 processing has not been disrupted or changed, instead, the transaction is queued by the message channel driver 50, allowing the task 1a processing to complete such that the next task, task 2, may be processed.

Meanwhile, in some embodiments, when the message channel transaction that was initiated during task 1a completes, a hardware event is generated, in time period 103 (block 400).
20 The message channel driver 50 responds to the hardware event by issuing a callback to the address of the second part of task 1, task 1b, in time period 104 (block 402). The callback address 40 for the given pcode flow (user) 30 contains the address. Task 1b commences in time period 105 (block 404). Since the message channel transaction is completed, the results processing is performed, in time period 106 (block 406), and task 1b ends, in time period 107
25 (block 407).

It is instructive to compare the operations 200 of Figure 3 with the optimized operations 300, 400 of Figure 4. In Figure 3, two pcode flow tasks, task 1 and task 2, are processed in 108 time periods. In Figure 4, the same two pcode flow tasks, task 1 and task 2, with the first task further being subdivided into tasks 1a and 1b, are processed in 107 time periods, for an
30 improvement of one time period. Furthermore, however, there is a further opportunity for additional tasks to be processed sequentially following the completion of task 2, with 95 time periods being available between time period 8 and time period 103. And, either tasks that issue message channel transactions (such as task 1) or tasks that do not issue message channel transactions (such as task 2) may be processed during these 95 time periods. Thus, the

operations 300, 400 of Figure 4 provide additional opportunities for efficient processing of message channel transactions, without the need to block one or more transactions.

Further, in some embodiments, the processing of the task 2 transactions commences much earlier with the method 300 than in the method 200. And, by being able to process more tasks sequentially in the 95 additional time periods, the cycles may provide a functional benefit to the system processing the transactions. In some embodiments, the pcode issues hundreds of message channel transactions per millisecond, so the potential benefit of using the cycles may be significant.

Recall that both the message channel driver 50 and the users 30 use tools 80 to assist with performing the optimization method 100. Figure 5 shows the tools 80 used by the optimization method 100, in some embodiments. There are four functions 72, 74, 76, and 78, which are classified as application programming interface (API) functions and two functions 82 and 84, which are deemed helper functions. Functionally, the user 30 interfaces with the driver 50 (via the API functions 72, 74, 76, 78), and the driver interfaces with the processor. The API functions, therefore, enable the users to access the processor, via the driver 50.

In some embodiments, the first API function 72, `is_virtual_message_channel_busy`, enables the user 30 to determine whether its virtual channel is busy or idle. The VCID 60 for the user 30 is provided as an input to this function 72. Recall that, based on the VCID 60, the virtual channel 70 is assigned to the user 30. If the user 30 senses a request on its virtual channel 70, another request cannot be sent until the first message has been processed. Thus, before proceeding with the second message, the user 30 ascertains, using the API function 72, whether its virtual message channel 70 is not busy. Further, if the message channel request is one that returns data, the user 30 uses the API function 72 to determine whether the request has been completed, thus enabling the user to retrieve the returned data. The API function 72 is thus essentially a handshaking mechanism between the user 30 and the message channel driver 50. The API function 72 returns a run/busy bit for the virtual message channel 70 specified by the VCID 60 for the user 30.

In some embodiments, the second API function 74, `message_channel_virtual_read`, returns the contents of the virtual message channel 70 specified by the input, VCID 60. In some embodiments, a 64-bit message channel payload result is returned by the API function 74. The message channel driver 50 returns the read information to the buffer 90 associated with the user 30. Thus, upon invoking this read function 74, the user 30 will read the contents of the buffer 90.

In some embodiments, the third API function 76, `message_channel_virtual_send`, is the means by which the user 30 sends a message on the message channel 20, using its virtual

message channel 70. Recall that the message channel 20 is used by many disparate pcode flows and functions between endpoints in the processor, such as to read and write uncore control registers, issue power management requests (PMReqs), as well as to send messages to other platform entities (e.g., other CPUs 150, PCHs 180 in the multiprocessor system 500). Again, the
5 VCID 60 for the user 30 is provided as an input to this function 76. Once the message is on the virtual message channel 70, the message channel driver 50 is able to process the message on the message channel 30.

In some embodiments, the fourth API function 78, `message_channel_virtual_send_PMreq`, is a special version of the third API function 76, which processes a PMreq transmission. As with
10 the other API functions, the VCID 60 for the user 30 is provided as an input to this function 78. PMreq, short for power management request, is a special type of transaction used by the QPI bus that interconnects between CPUs (Figure 1).

While both the users 30 and the message channel driver 50 use the above API functions, only the driver 50 uses the helper functions 82 and 84, in some embodiments. The helper
15 functions allow facile movement of data between the virtual channels 70 and the physical message channel 20. The first helper function, `message_channel_send_to_physical` 82, enables the message channel driver 50 to send data to the physical message channel 20. The second helper function, `message_channel_poll_physical_with_timeout` 84, enables the message channel driver 50 to poll the physical message channel for completion of an operation, and includes a
20 timeout.

In some embodiments, the message channel driver 50 registers an event that causes a kernel within the single processor-based system or the multiprocessor system to run a callback function when the message channel transaction has completed. The event is a hardware event, such as an interrupt, that indicates that the message channel 20 is no longer busy, which means
25 that the last thing the driver put in the message channel has completed.

In some embodiments, on behalf of a user (pcode flow) 30, the message channel driver 50 puts a message from the assigned virtual channels 70 of the user into the message channel 20. The message channel driver 50 understands whether the message is to return data or not. Where data is to be returned, the message channel driver 50 retrieves the data and puts it in the buffer 90
30 dedicated to the user 30. Subsequent to the retrieval by the driver 50, the user 30 employs API read function 74 to retrieve the contents of the buffer 90.

Once the processing of a message channel transaction on behalf of one user is completed, the message channel driver 50 may proceed to “connect” another virtual channel 70 to the message channel 20 on behalf of another user 30.

Figures 6 and 7 are flow diagrams showing operations of the message channel optimization method 100 in processing a send operation and a read operation, respectively, according to some embodiments. Figure 6 illustrates a send operation by one of the users 30 of either a single processor-based system or a multiprocessor system, such as the multiprocessor system 500 of Figure 1, while Figure 7 illustrates a read operation.

First, Figure 6 describes the send operation. Before the user 30 can issue any transaction on the message channel 20, the user-assigned virtual message channel 70 must be available, in some embodiments. Thus, the user issues the first API function 72 to determine whether the virtual message channel 70 is available (block 102). Once available, the user 30 issues an API send function, either the general API send function 76 or the specialized API send PMreq function 78 (block 104). The user 30 also registers a callback address (block 106) that the message channel driver 50 will use to return to a predetermined address of the user (block 106). At this point, the message channel (send) transaction is queued by the message channel driver 50 (block 108). This frees up the user 30 to continue with other transaction processing.

When the send transaction is issued by the user 30, the message channel 20 may not be available. The pcode of a typical multiprocessor system uses the message channel quite frequently, from hundreds to thousands of times per millisecond. Thus, until the message channel 20 is not busy, the send transaction is not processed (block 110). Once the message channel 20 is available, a hardware event, such as an interrupt, will notify the message channel driver 50 that the message channel is available.

The message channel driver 50, however, is processing virtual message channel transactions for a number of different users. Once the message channel 20 is available and once the user is at the top of the message channel driver's queue 110, the message channel driver 50 sends the message channel transaction from the user's virtual channel 70 to the message channel 20 (block 112). Once at the message channel 20, the message channel transaction is processed (block 114). The message channel driver 50 then registers an event. This event causes the kernel of the single processor-based system or multiprocessor system to run the callback function to the callback address 40 of the user 30 (block 116). The user 30 is thus able to complete the send transaction (block 118).

Figure 7 shows similar operations, this time where the user 30 is transmitting a read operation to the message channel 20. Again, the user 30 first ensures that the virtual channel 70 is available (block 152). Once available, the user 30 issues the API read function 74 to its virtual message channel 70 (block 154). The user also registers its callback address (block 156) so that the message channel driver 50 will be able to get back to the user pcode flow 30 once the

message channel transaction is complete. The message channel driver 50 queues the transaction (block 158).

Once the message channel 20 is available (block 160) and once the user is at the top of the message channel driver's queue 110, the message channel driver 50 sends the message channel transaction from the user's virtual channel 70 to the message channel 20 (block 162). Once at
5 the message channel 20, the message channel transaction is processed (block 164). The message channel driver 50 then registers an event. This causes the kernel of the single processor-based system or multiprocessor system to run the callback function to the callback address 40 of the user 30 (block 166). Since the transaction is a read transaction, the message channel driver 50
10 put the contents of the read in the buffer 90 of the user 30. The user 30 thus retrieves the contents of the buffer 90 (block 168) and completes the read transaction (block 170).

The message channel optimization method and system 100 thus enable multi-flow access to the message channel infrastructure within a CPU of a single processor-based system or multiprocessor system. The user employs a virtual channel to submit message channel
15 transactions, with the message channel driver processing the transaction "behind the scenes". Each transaction will be processed, either immediately or at some future time, by the message channel driver. Tasks involving message channel transactions as well as non-message channel transactions are processed more efficiently, in some embodiments.

While the application has been described with respect to a limited number of embodiments,
20 those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of the invention.

We claim:

1. A non-transitory computer-readable medium including code, when executed, to cause a machine to perform the following operations:

issue a message channel transaction to an assigned virtual message channel, wherein the message channel transaction is queued by a message channel driver;

5 queue the message channel transaction;

process a succeeding transaction following the message channel transaction without waiting for the message channel transaction to be completed;

receive a hardware event to indicate that a physical message channel is available; and

10 transfer the message channel transaction from the assigned virtual message channel to the physical message channel in response to receiving the hardware event.

2. The non-transitory computer-readable medium of claim 1, the code, when executed, to cause the machine to further perform the following operations:

indicate, by way of a virtual message channel identifier, which of the plurality of virtual message channels is available.

15 3. The non-transitory computer-readable medium of claim 1, the code, when executed, to cause the machine to further perform the following operations:

indicate a return address to be transferred to after transferring the message channel transaction to the physical message channel.

20 4. The non-transitory computer-readable medium of claim 1, the code, when executed, to cause the machine to further perform the following operations:

retrieve data from the physical message channel; and

store the data in a buffer.

5. A method to process message channel transactions without performing blocking operations, the method comprising:

25 issuing, by a user, a message channel transaction to a message channel, wherein the user comprises a portion of portable machine code (pcode) that is executed by a processor in a processor-based system and the message channel transaction is received into a virtual message channel assigned to the user;

30 queueing, by a driver, the message channel transaction, wherein the user is able to further process pcode without waiting for the message channel transaction to complete;

receiving, by the driver, a hardware event indicating that the message channel is available;

transferring, by the driver, the message channel transaction of the user from the virtual message channel to the message channel, wherein the message channel transaction is carried by the message channel to an endpoint;

returning, by the driver, to the user pcode using a callback address provided by the user; and

completing, by the user, execution of the message channel transaction.

6. The method of claim 5, further comprising:

issuing, by the user, a function to the driver, wherein the function tells the user whether the virtual message channel is busy or not.

10 7. The method of claim 5, further comprising:

issuing, by the driver, an event to the processor, wherein the event causes a kernel to run a callback function to the callback address.

8. The method of claim 5, further comprising:

storing, by the driver, message channel transaction results stored in a buffer; and

15 retrieving, by the user, the results from the buffer.

9. A method to process two tasks in a processor-based system, the two tasks being executed as part of a portable machine code (pcode) executed by the processor-based system, the method comprising:

20 executing a data preparation portion of a first task, the first task comprising the data preparation, issuance of a message channel transaction, and results processing;

sending a function to a processor in the processor-based system, wherein the function causes the message channel transaction to be queued by a message channel driver; and

25 executing a second task, the second task comprising no message channel transaction, wherein the second task is executed irrespective of whether the message channel transaction of the first task is completed;

wherein the message channel driver, upon receiving a hardware event, transfers the message channel transaction from a virtual message channel to a message channel where the message channel transaction is carried by the message channel to an endpoint.

10. The method of claim 9, further comprising:

executing the results processing portion of the first task.

11. The method of claim 9, further comprising:

sending a callback address to the message channel driver;

wherein the message channel driver calls the callback address after the message channel
5 transaction is executed.

12. An apparatus, comprising:

transaction means to initiate a message channel transaction to a virtual message channel
and to register a callback address;

driver means to queue the message channel transaction by the user and, in response to the
10 message channel being available, to transfer the message from the virtual message channel to the
message channel;

channel means to process the message;

callback means to cause the driver means to register an event to cause a kernel to run a
callback function to the callback address; and

15 completion means to complete the message channel transaction.

13. The apparatus of claim 12, further comprising:

a first means to determine whether the virtual message channel is busy;

the driver means to indicate whether the virtual message channel is busy upon receiving the first
means.

20 14. The apparatus of claim 13, further comprising:

a second means to request a read of the virtual message channel;

the driver means to return contents of the virtual message channel to a buffer upon receiving the
second means.

15. The apparatus of claim 14, further comprising:

a third means to request a send of data stored in the virtual message channel to the
message channel;

25 the driver means to transfer the data from the virtual message channel to the message channel
upon receiving the third means.

16. The apparatus of claim 12, further comprising:

a fourth means to transfer data from the virtual message channel to the physical message channel.

17. An apparatus comprising:

processing logic to process a first task;

5 channel logic to issue a first message channel transaction in response to the processing logic to process the first task;

message channel driver logic to hold a reference to the first message channel transaction;

wherein the processing logic is further to start processing a second task before completion of the first message channel transaction; and

10 wherein the message channel driver logic is further to initiate subsequent processing of the first task subsequent to the processing logic to start processing of the second task in response to an event associated with completion of the first message channel transaction.

18. The apparatus of claim 17, further comprising:

a queue to hold the reference to the first message channel transaction.

15 19. The apparatus of claim 17, further comprising:

a plurality of tools to be used by the message channel driver, the plurality of tools comprising:

a busy function to determine whether the channel logic is available or busy;

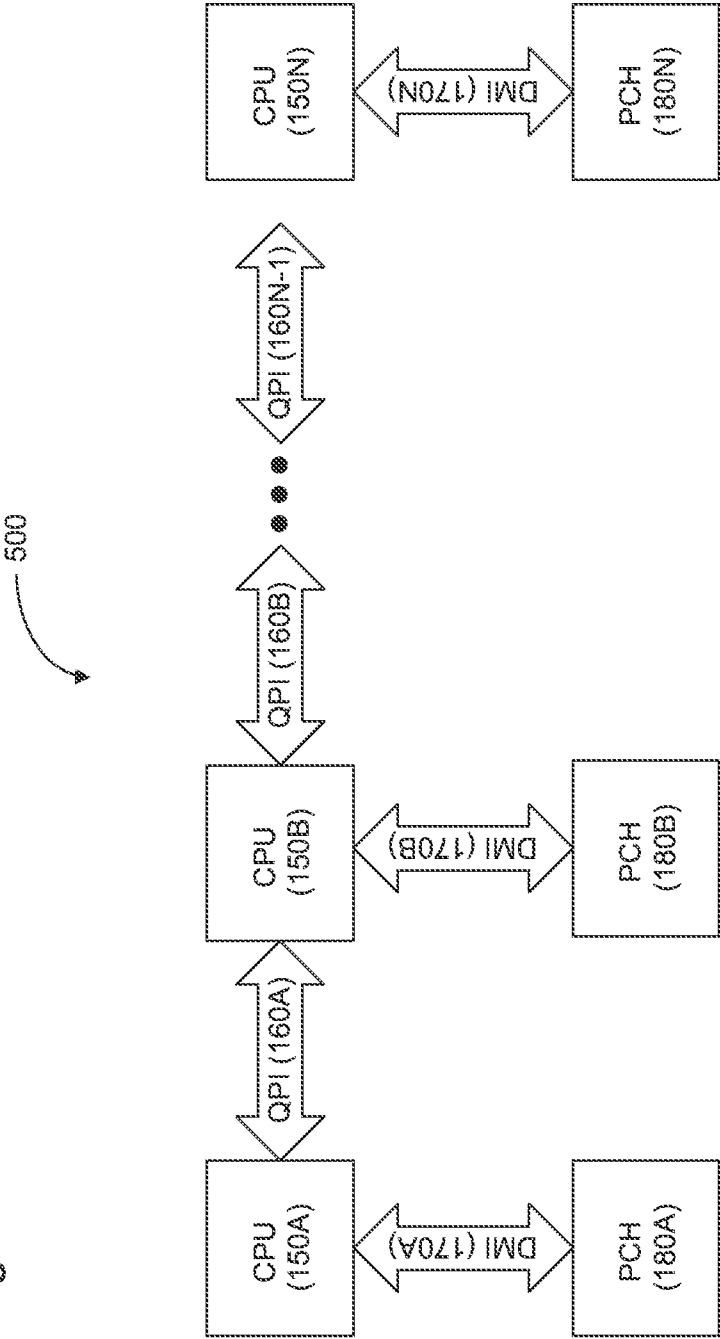
a read function to read the contents of the channel logic; and

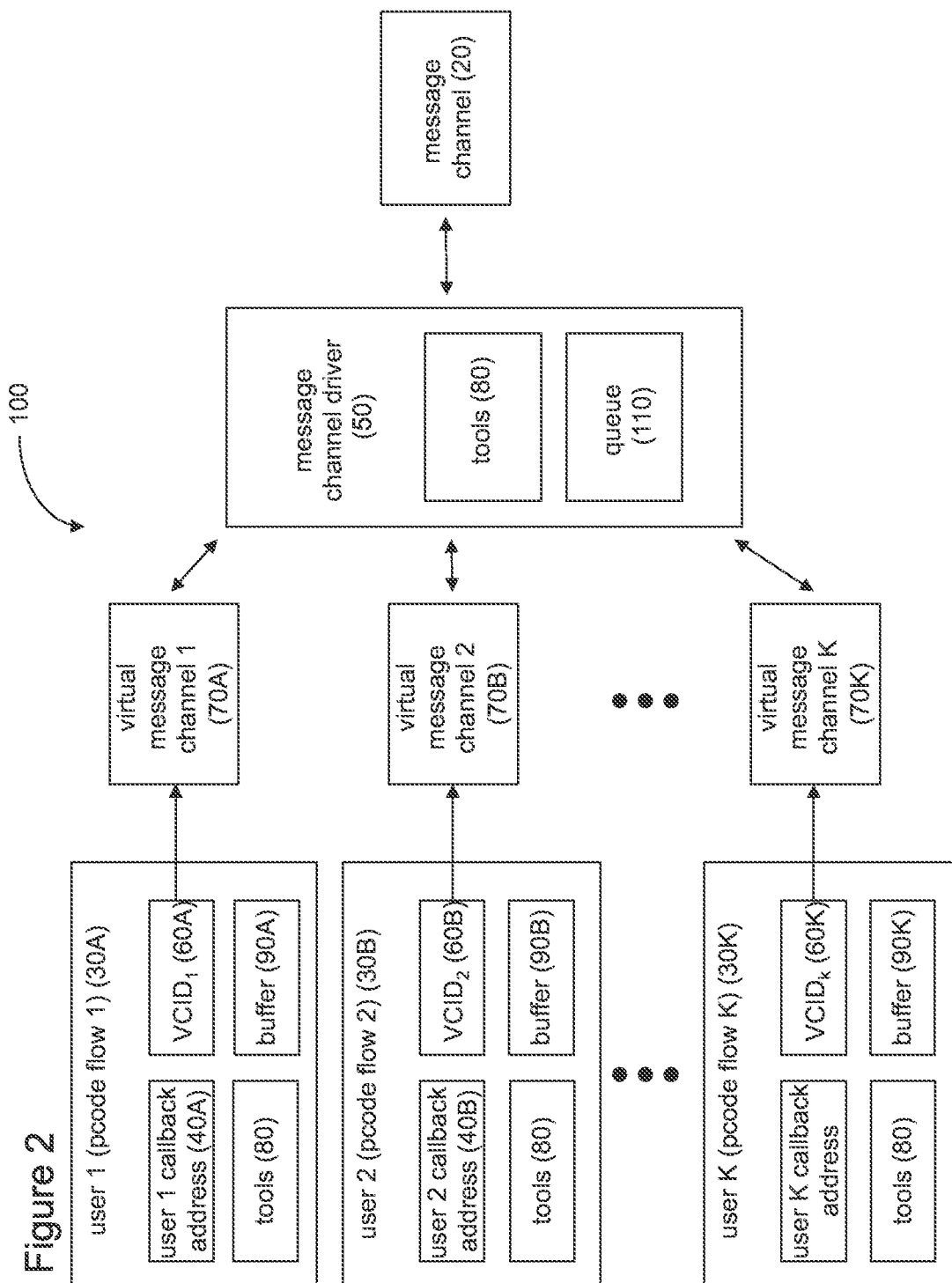
20 a send function to send data to the channel logic.

20. The apparatus of claim 19, the plurality of tools further comprising:

a send power management request function to send a power management request to the channel logic.

Figure 1





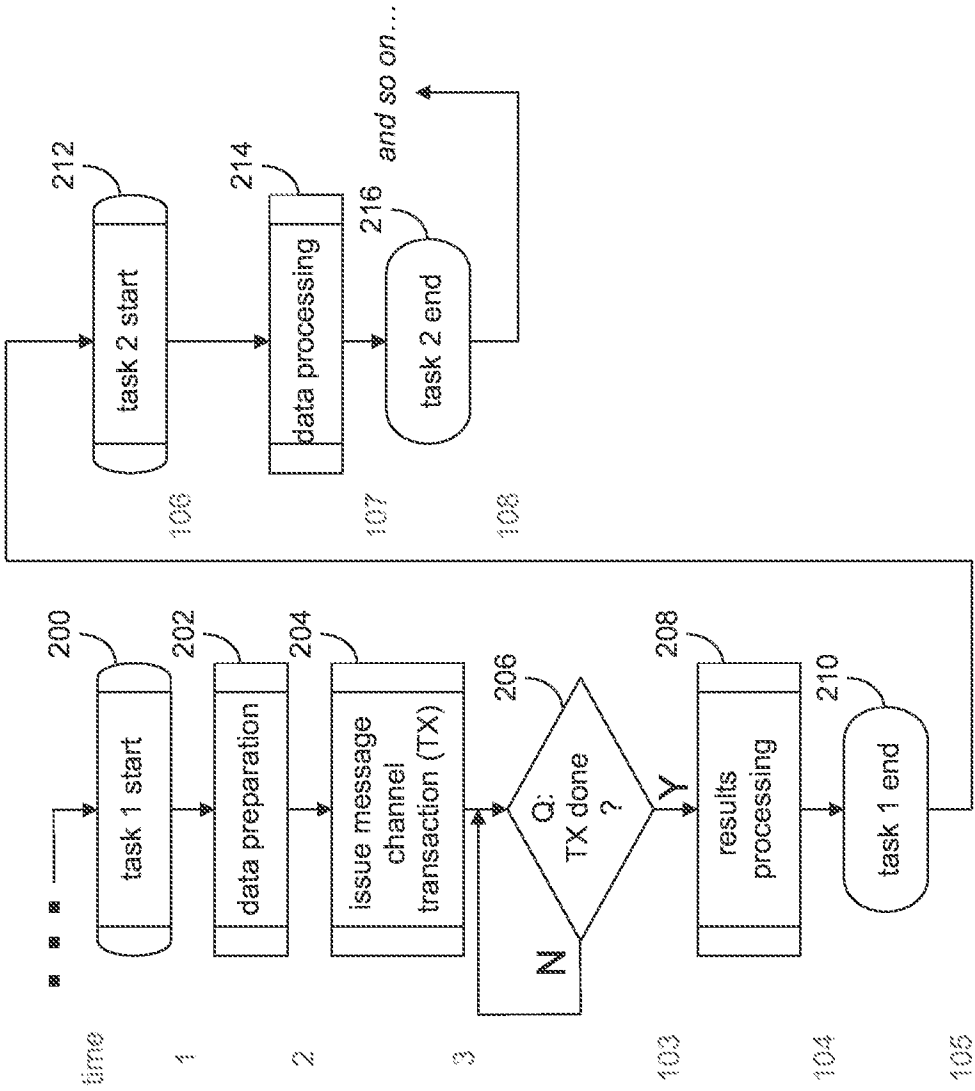


Figure 3

4. Engineering

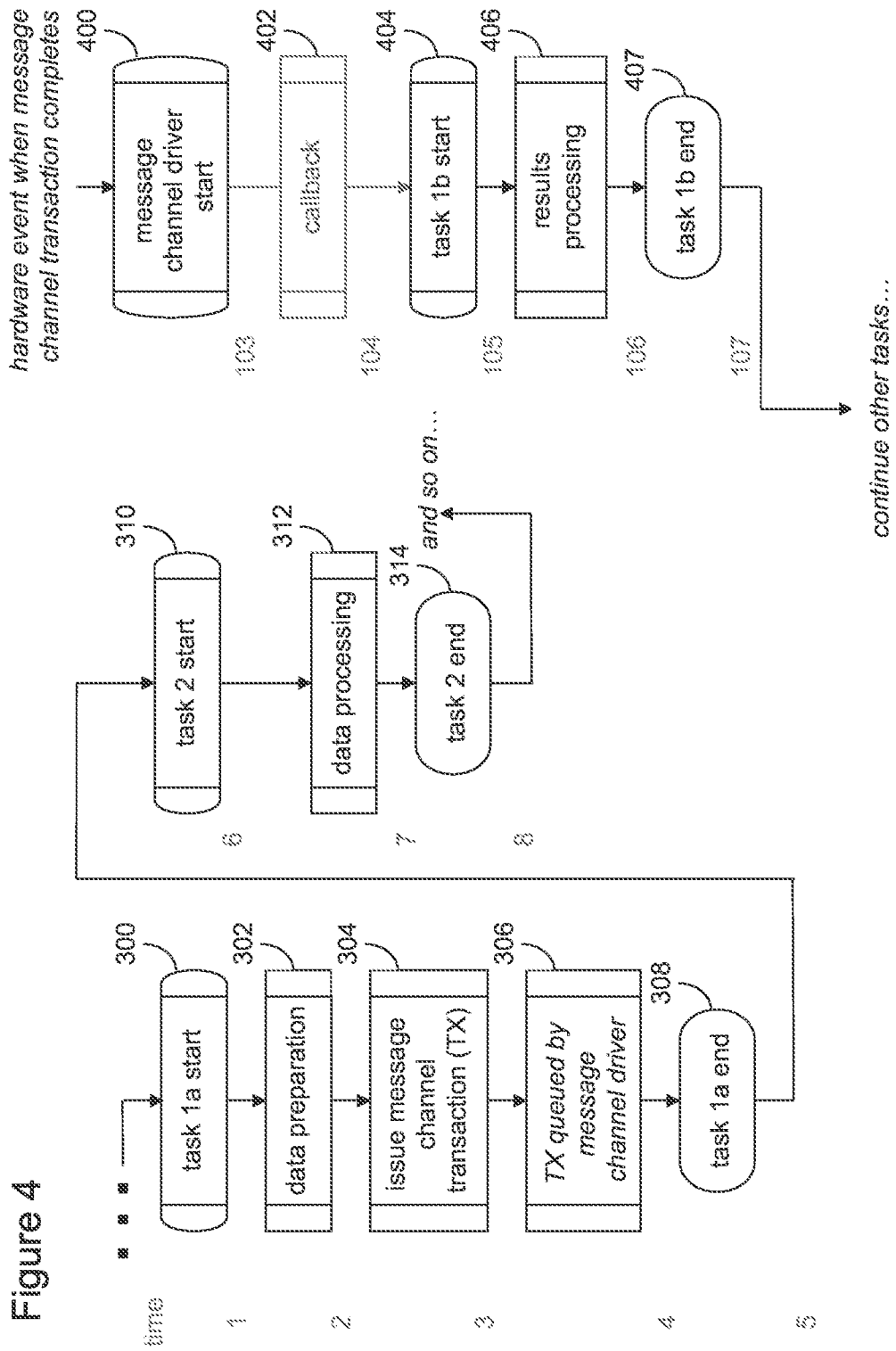
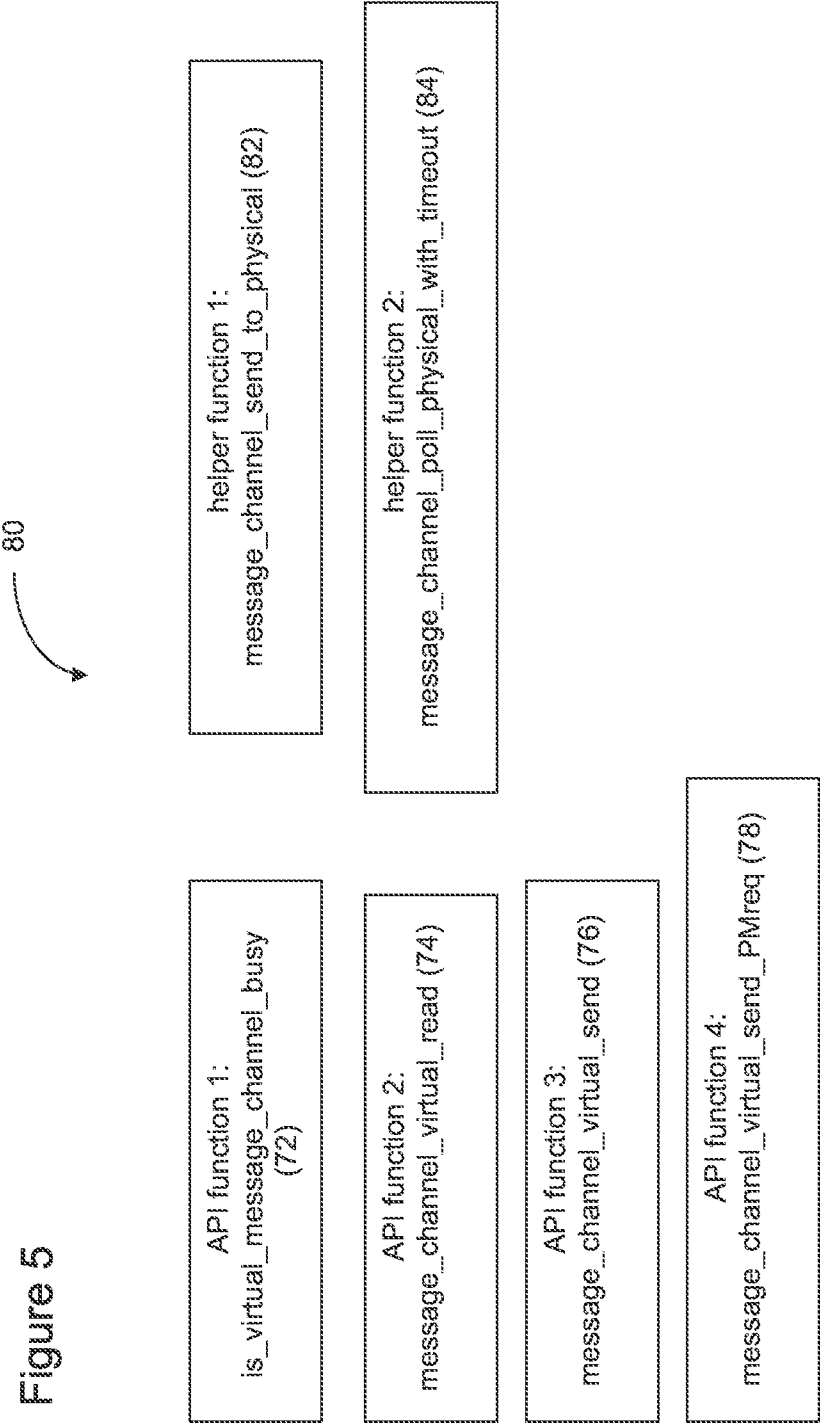
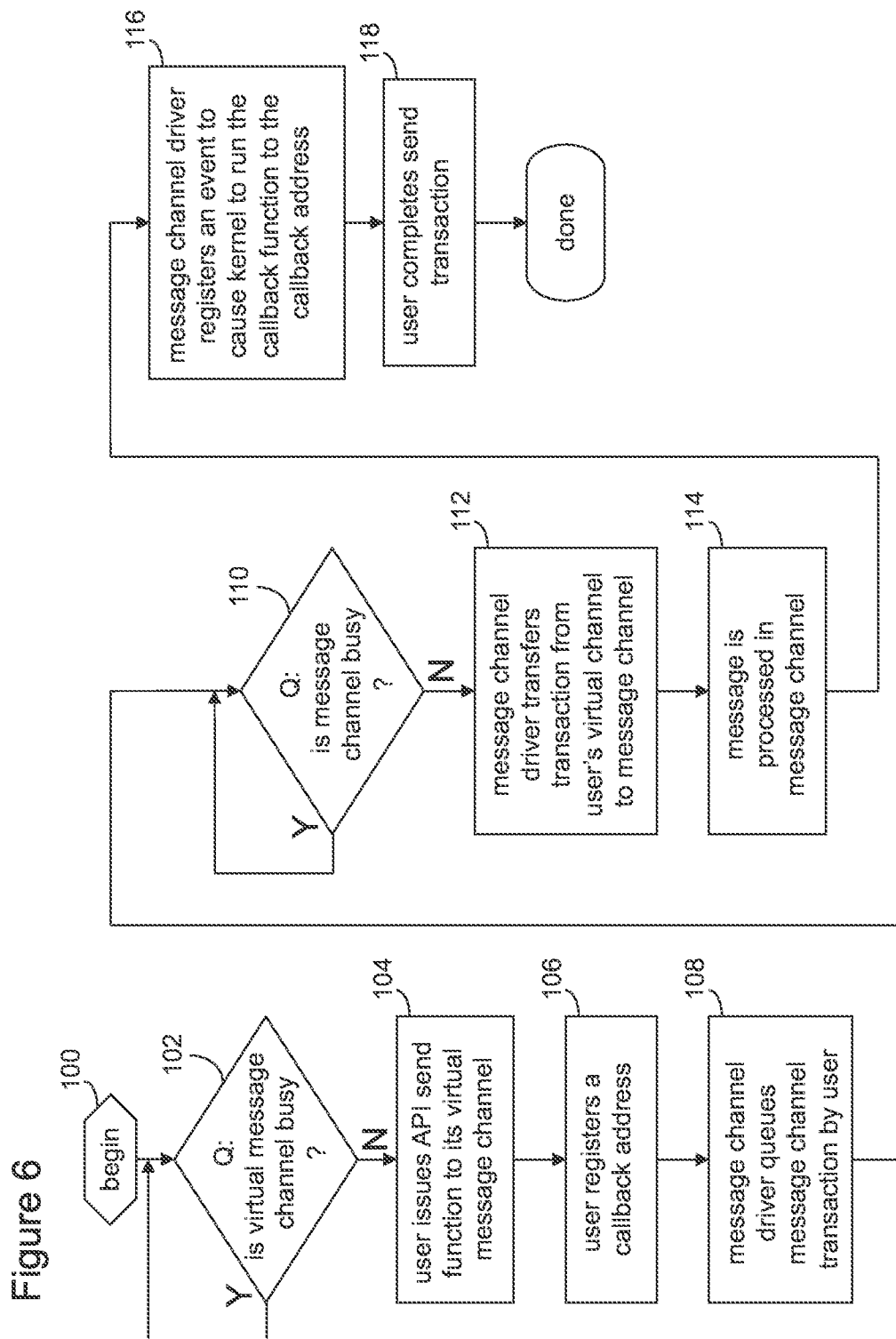
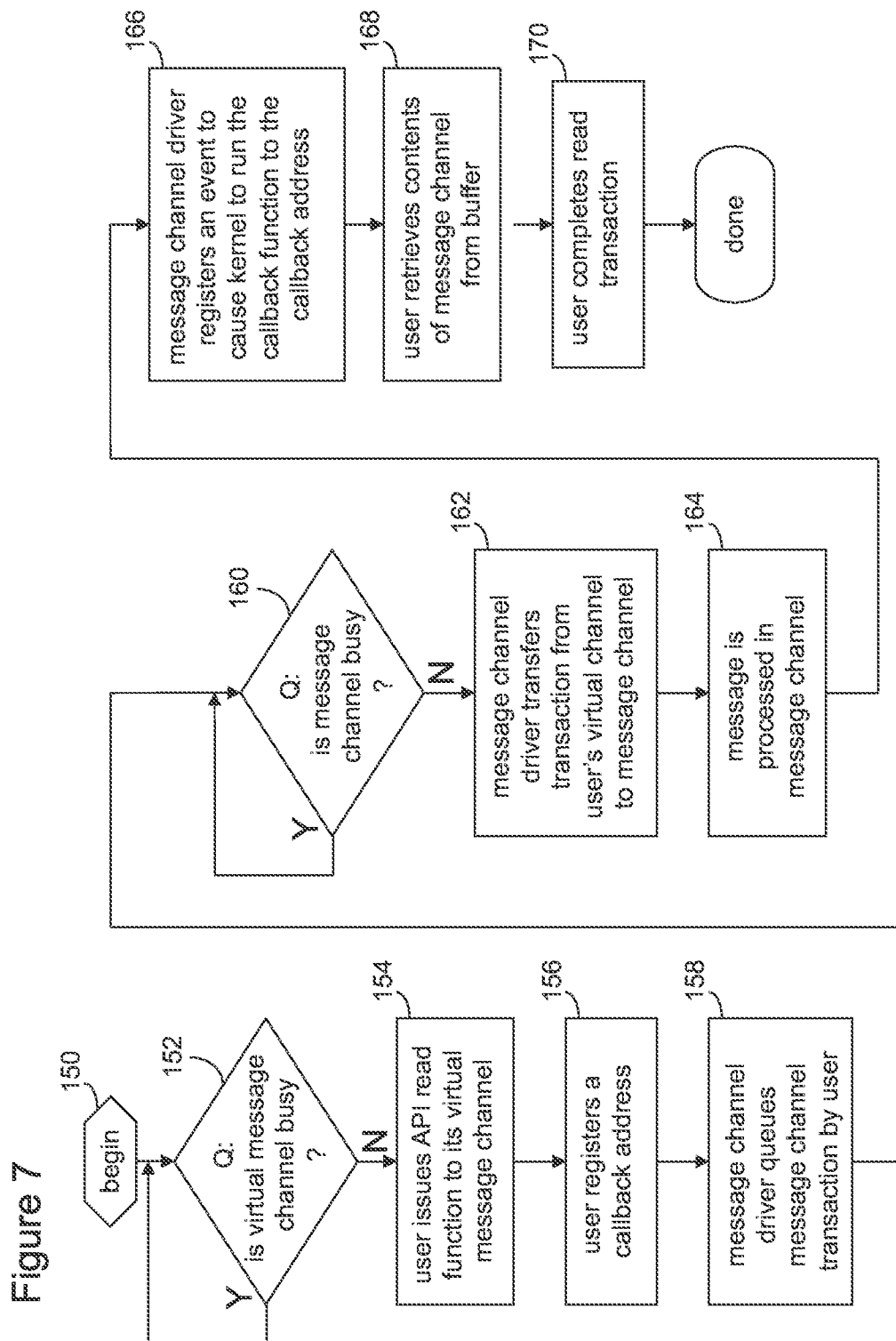


Figure 5







A. CLASSIFICATION OF SUBJECT MATTER**G06F 13/14(2006.01)i, G06F 13/36(2006.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 13/14; G06F 15/16; G06F 15/173; H04L 12/00; H04Q 11/00; G06F 9/46; G06F 9/38; G06F 13/36

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & keywords: processor, virtual message channel, code, physical message channel, queue, second task, callback address, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2003-0056020 A1 (SHAILENDER CHAUDHRY et al.) 20 March 2003 See paragraphs 39-44, 49-53, 60; figures 1-2, 6; and claim 1.	1-20
A	WO 2011-053891 A2 (RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY) 05 May 2011 See paragraphs 17-29; figures 1-4; and claim 1.	1-20
A	US 6055618 A (GREGORY M. THORSON) 25 April 2000 See column 7, lines 17-35; figure 5; and claim 1.	1-20
A	US 2007-0053350 A1 (AARON T. SPINK et al.) 08 March 2007 See paragraphs 11-30; and figure 1.	1-20
A	US 2010-0017513 A1 (DENNIS C. ABTS) 21 January 2010 See paragraphs 17-31; figures 2-3; and claims 1-3.	1-20



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

22 January 2014 (22.01.2014)

Date of mailing of the international search report

23 January 2014 (23.01.2014)

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City,
302-701, Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

BYUN, Sung Cheal

Telephone No. +82-42-481-8262



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2013/063647

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003-0056020 A1	20/03/2003	US 7152232 B2	19/12/2006
WO 2011-053891 A2	05/05/2011	US 2012-0324462 A1	20/12/2012
		WO 2011-053891 A3	13/10/2011
US 06055618 A	25/04/2000	EP 0858633 A1	24/02/1999
		EP 0858633 B1	24/11/1999
		JP 03639848 B2	20/04/2005
		JP 03933620 B2	20/06/2007
		JP 2000-508094 A	27/06/2000
		JP 2004-118855 A	15/04/2004
		WO 97-16792 A1	09/05/1997
US 2007-0053350 A1	08/03/2007	US 7643477 B2	05/01/2010
US 2010-0017513 A1	21/01/2010	US 2012-0265883 A1	18/10/2012