

Exhibit 596-17

Invalidity of the '596 Patent by Van Loo '485 and Other Prior Art

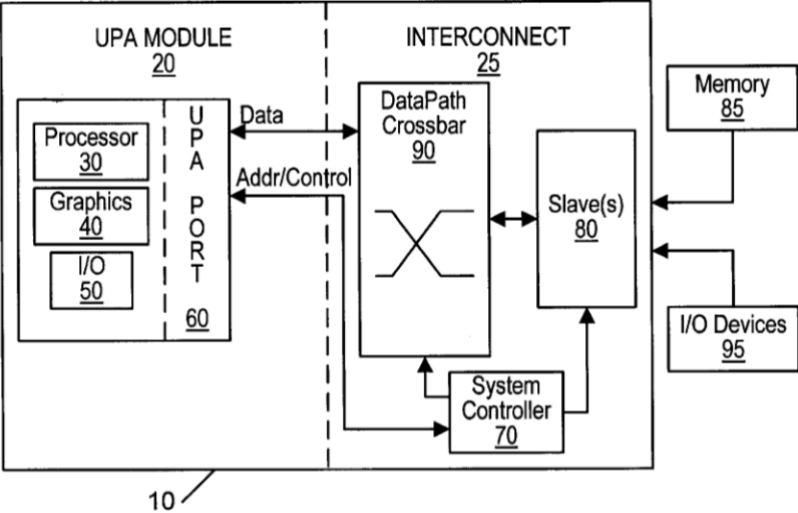
The asserted claims of U.S. Patent No. 8,924,596 are anticipated and/or rendered obvious in view of US5907485A ("Van Loo '485") alone or in combination with knowledge of one of skill in the art or other prior art.

This claim chart is based on Defendants' present understanding of the asserted claims, Plaintiffs' vague and overbroad infringement contentions (and corresponding, implicit claim constructions), and Defendants' investigation to date. Defendants are not adopting Plaintiffs' apparent constructions, nor are Defendants admitting to the correctness of any particular construction. Defendants reserve all rights to amend this invalidity claim chart in light of any claim construction developments or any amendments to Plaintiffs' infringement contentions. Further, as discovery is ongoing and Defendants continue to seek discovery from third parties regarding the references identified in Defendants' invalidity contentions as well as other potential prior art, Defendants reserve the right to amend or supplement this claim chart in connection with Defendants' continuing investigation. At a minimum, Defendants intend to revise these invalidity contentions as appropriate based on discovery from third parties as discovery progresses.

The claim chart below identifies where each limitation of each asserted claim of the '596 patent can be found in Van Loo '485. This analysis is based on Plaintiff's interpretation of the claims as indicated in their Infringement Contentions. The citations provided below are exemplary, rather than exhaustive, and Defendants reserve the right to rely upon any other portion of the cited references.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[1pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	To the extent the preamble is considered a claim limitation, Van Loo '485 teaches "[a] system for dividing and synchronizing a processing task across a plurality of processing elements," including at least the following exemplary citations. For example, Van Loo '485 teaches a system for dividing and synchronizing processing tasks by disclosing, as non-limiting examples, master devices, slave devices, system controllers, instruction and output queues that coordinate, divide, and synchronize transaction requests. <i>See, e.g.:</i> "This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full." Van Loo '485 at Abstract.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1 Van Loo '485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture centered around a DataPath Crossbar (92). On the left, three Master blocks (M1, M2, M3) are shown, each with the note "(can act as Slave)". They are connected to the crossbar via data lines D1, D2, and D3. A boot PROM (94) is also connected to the crossbar via data line D4. On the right, two Memory blocks (Mem1, Mem2) and two Slave blocks (S1, S2) are connected to the crossbar via data lines D4, D5, D6, and D7. A System Controller (75) is at the bottom, connected to the crossbar via a "DataPath Control" line. It also manages several address/control lines: A/ctrl-1, A/ctrl-2, A/ctrl-3, A/ctrl-4, A/ctrl-5, A/ctrl-6, and A/ctrl-7. Dashed lines indicate that the masters and slaves can also function as slaves to other masters. FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Contains Logic 110, Logic 112, MASTER QUEUE 290, Config Register 270, and Counter 310. Master 2 (125): Contains Logic 122, MASTER QUEUE 300, Config Register 280, and Counter 320. Slave 1 (140): Contains Logic 142, Slave Queue 185, ID Register 188, and Counter 189. Slave 2 (150): Contains Logic 152, Slave Queue 195, ID Register 190, and Counter 191. System Controller (130): Contains SC LOGIC 245, SCID Register 260, SCID Register 250, SCID Register 255, SCID Register 258, SCID Register 265, SCID Register 270, SCID Register 275, SCID Register 280, SCID Register 285, SCID Register 290, SCID Register 295, SCID Register 300, SCID Register 305, SCID Register 310, SCID Register 315, SCID Register 320, SCID Register 325, SCID Register 330, SCID Register 335, SCID Register 340, SCID Register 345, SCID Register 350, SCID Register 355, SCID Register 360, SCID Register 365, SCID Register 370, SCID Register 375, SCID Register 380, SCID Register 385, SCID Register 390, SCID Register 395, SCID Register 400, SCID Register 405, SCID Register 410, SCID Register 415, SCID Register 420, SCID Register 425, SCID Register 430, SCID Register 435, SCID Register 440, SCID Register 445, SCID Register 450, SCID Register 455, SCID Register 460, SCID Register 465, SCID Register 470, SCID Register 475, SCID Register 480, SCID Register 485, SCID Register 490, SCID Register 495, SCID Register 500, SCID Register 505, SCID Register 510, SCID Register 515, SCID Register 520, SCID Register 525, SCID Register 530, SCID Register 535, SCID Register 540, SCID Register 545, SCID Register 550, SCID Register 555, SCID Register 560, SCID Register 565, SCID Register 570, SCID Register 575, SCID Register 580, SCID Register 585, SCID Register 590, SCID Register 595, SCID Register 600, SCID Register 605, SCID Register 610, SCID Register 615, SCID Register 620, SCID Register 625, SCID Register 630, SCID Register 635, SCID Register 640, SCID Register 645, SCID Register 650, SCID Register 655, SCID Register 660, SCID Register 665, SCID Register 670, SCID Register 675, SCID Register 680, SCID Register 685, SCID Register 690, SCID Register 695, SCID Register 700, SCID Register 705, SCID Register 710, SCID Register 715, SCID Register 720, SCID Register 725, SCID Register 730, SCID Register 735, SCID Register 740, SCID Register 745, SCID Register 750, SCID Register 755, SCID Register 760, SCID Register 765, SCID Register 770, SCID Register 775, SCID Register 780, SCID Register 785, SCID Register 790, SCID Register 795, SCID Register 800, SCID Register 805, SCID Register 810, SCID Register 815, SCID Register 820, SCID Register 825, SCID Register 830, SCID Register 835, SCID Register 840, SCID Register 845, SCID Register 850, SCID Register 855, SCID Register 860, SCID Register 865, SCID Register 870, SCID Register 875, SCID Register 880, SCID Register 885, SCID Register 890, SCID Register 895, SCID Register 900, SCID Register 905, SCID Register 910, SCID Register 915, SCID Register 920, SCID Register 925, SCID Register 930, SCID Register 935, SCID Register 940, SCID Register 945, SCID Register 950, SCID Register 955, SCID Register 960, SCID Register 965, SCID Register 970, SCID Register 975, SCID Register 980, SCID Register 985, SCID Register 990, SCID Register 995. Connections include Data paths between Master 1 and Slave 1, Master 2 and Slave 2, and a DataPath Control block (155) connecting to the Slave Queues. The System Controller is connected to the Slave Queues and I/O Devices & Memory (95). 100 FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a memory system architecture. On the left, two UPA ports are shown: UPA port₀ (630) and UPA port₃₁ (640). Each port contains a Dtag block. UPA port₀ is connected to a UPA_Databus (632) and sends a P_REQ signal (1) to a Datapath Crossbar Nxm (625). The crossbar receives S_REPLY (2) and P_REPLY (2) signals and outputs data (3) to Memory Bank₀ (610) and Memory Bank_{m-1} (620). The crossbar is also connected to a System Controller & Address Crossbar (650) via a Datapath Control line (660). The system controller is connected to the crossbar and provides Memory Control (670) to the memory banks. A Snoobus (600) is connected to the Dtag blocks and the system controller. A Memory Write Flow Control (600) is also shown. Various control signals are indicated, including Addr_Valid, Data_Stall, ECC_Valid, Arb_Cntst, Port_ID, and Sys_Clk. FIG. 5 Van Loo '485 at Figure 5.

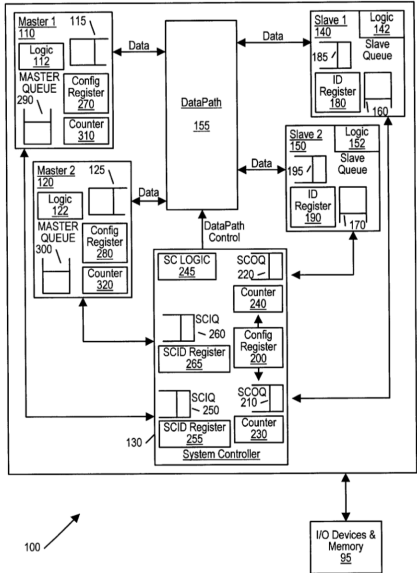
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 – col. 3, l. 6. “FIG. 1 is a top-level block diagram of a computer system 10 incorporating the present invention. This diagram relates to a specific implementation of applicant's new Ultrasparc Architecture, which is described fully in the document UPA Interconnect Architecture, by Bill van Loo, Satya Nishtala and Zahir Ebrahim. Sun Microsystems, Inc.'s internal release version 1.1 of the UPA Interconnect Architecture has been

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	submitted as Appendix A to a related patent application by applicant, entitled "Method and Apparatus for Flow Control in a Packet-Switched Computer System", by Ebrahim et al. That patent application, filed in the United States Patent Office on Mar. 31, 1995, describes many of the broader features of the UPA architecture, and is incorporated herein by reference.” Van Loo ’485 at col. 4, ll. 12–25. “The present invention uses a new system interconnect architecture and concomitant new methods for utilizing the interconnect to control transaction requests and data flow between master devices and slave or memory devices. In FIG. 1, the system 10 includes a UPA module 20 and an interconnect network or module 25, which in different embodiments of the invention may or may not be connected to the data path. The UPA module may include such devices as a processor 30, a graphics unit 40, and an I/O unit 50. Other units may be included, and act as the master units for the purposes of the present invention. A master interface is defined as the interface for any entity initiating transaction requests; examples of such masters are a CPU making memory requests, or an I/O channel and bridges making DMA requests.” Van Loo ’485 at col. 4, ll. 27–41. “FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[1a] an input queue implemented in hardware;	Van Loo ’485 teaches “an input queue implemented in hardware,” including at least the following exemplary citations. For example, Van Loo ’485 teaches an input queue implemented in hardware, by disclosing, as non-limiting examples, a slave queue. <i>See, e.g.</i>: Defendants have proposed a construction that “queue implemented in hardware” means “queue implemented in physical circuits without software intervention.” The input queue in Van Loo ’485 meets Defendants’ construction because, for example, Van Loo ’485 teaches a hardware implementation of its teachings, which includes its teachings of queues. <i>See, e.g.</i>, Van Loo ’485 at 7:50–53 (“The particular implementation of these logic modules [including the slave receive queues], and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2.”); 10:30–32 (“The logic for the carrying out of the invention is provided by hardware/firmware logic of the SC ASIC and the master and slave interfaces, and by program instructions stored in memory 85 of the system, as shown in FIG. 1.”). “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo ’485 at Abstract.  FIG. 2 Van Loo ’485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a memory system architecture. On the left, two UPA ports are shown: UPA port₀ (630) and UPA port₃₁ (640). Each port contains a Dtag block. UPA port₀ is connected to a UPA_Databus (632) and sends a P_REQ signal (1) to a Datapath Crossbar Nxm (625). The Datapath Crossbar Nxm (625) is connected to a UPA_Addressbus (626) and a Memory_Databus (690). The Memory_Databus (690) is connected to a series of Memory Banks (610, 620, ..., 610). A System Controller & Address Crossbar (650) is connected to the UPA_Addressbus (626) and the Memory_Databus (690). The System Controller & Address Crossbar (650) also controls a Coherence/Memory/Datapath Control block (660). The Coherence/Memory/Datapath Control block (660) is connected to the UPA_Addressbus (626) and the Memory_Databus (690). A Memory Write Flow Control block (600) is connected to the UPA_Addressbus (626) and the Memory_Databus (690). A Snoobus is connected to the Dtag blocks of UPA port₀ and UPA port₃₁. A Diag Control block is connected to the Dtag blocks of UPA port₀ and UPA port₃₁. A Memory Control block (670) is connected to the Memory Banks (610, 620, ..., 610). A RAS/CAS block (670) is connected to the Memory Banks (610, 620, ..., 610). A Datapath Control block (680) is connected to the Datapath Crossbar Nxm (625) and the System Controller & Address Crossbar (650). A Dtag1 and Dtag2 block are connected to the Snoobus. A Memory Write Flow Control block (600) is connected to the UPA_Addressbus (626) and the Memory_Databus (690). FIG. 5 Van Loo '485 at Figure 5.

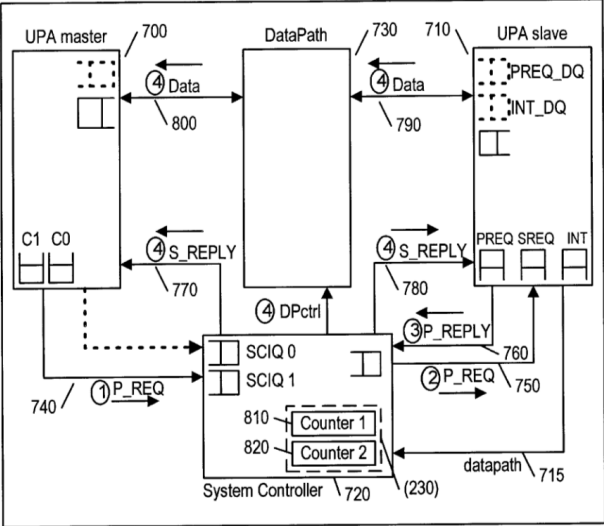
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 – col. 3, l. 6. “Each slave 140 and 150 includes a slave queue (160 and 170, respectively) for receiving transaction requests. The maximum sizes of these slave queues are represented by values in port ID registers 180 and 190, respectively.” Van Loo '485 at col. 6, ll. 32–35.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“Masters 110 and 120 include data queues or buffers 115 and 125, and slaves 140 and 150 include data queues or buffers 185 and 195, whose functions are described in detail relative to FIGS. 6 and 7 below. The maximum sizes of the slave write data queues 185 and 195 are also represented by values in port ID registers 180 and 190, respectively. In the special case where there is a one-to-one correspondence to a request queue entry (e.g. 100) and a data buffer in the write data queue (e.g. 185), with the write data queue being maximally dimensioned to hold an entire packet (i.e. dimensioned such that it can hold the largest contemplated packet size), then the queue size in 180 can be represented by a single number.” Van Loo ’485 at col. 6, ll. 36–48. “Many combinations of standard hardware and software are possible to execute these functions in the system controller; and the same is true of the functions carried out in the slave devices (see slave logic modules 142 and 152) and the master devices (see master logic modules 112 and 122). Here, the logic modules represent all of the circuitry, programming, memory and intelligence necessary to carry out the functions of the invention as described; assembling the hardware and software to do so is a matter of routine to one skilled in the art, given the teaching of this invention. (Where a master device is a processor, the logic for implementing the present invention can of course be made up in large part of the processor itself and the instructions it executes.) The particular implementation of these logic modules, and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2.” Van Loo ’485 at col. 7, ll. 38–54. “Any master can request transactions to any slave via the SC, for any selected number of master and slaves. The SC will typically be an ASIC configured for a given system with a predetermined maximum number of master and slave interfaces. Since it is a relatively simple and inexpensive ASIC (by comparison with the system as a whole), it provides great flexibility and economy by allowing designers to easily configure different SC's at low cost for different systems, each one tailored to the specific needs of that system.” Van Loo ’485 at col. 10, ll. 21–30. “The logic for the carrying out of the invention is provided by hardware/firmware logic of the SC ASIC and the master and slave interfaces, and by program instructions stored in memory 85 of the system, as shown in FIG. 1. Alternative embodiments may implement the logic in other fashions, e.g. by providing memories and general purpose processors to carry out any of the steps executed by the master interfaces, system controller and slave interfaces of the preferred embodiment of this invention.” Van Loo ’485 at col. 10, ll. 31–39. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[1b] an output queue implemented in hardware;	Van Loo ’485 teaches “an output queue implemented in hardware,” including at least the following exemplary citations. For example, Van Loo ’485 teaches an output queue implemented in hardware, by disclosing, as non-limiting examples, a master transaction request output queue or a system controller output queues. <i>See, e.g.:</i> Defendants have proposed a construction that “queue implemented in hardware” means “queue implemented in physical circuits without software intervention.” The output queue in Van Loo ’485 meets Defendants’ construction because, for example, Van Loo ’485 teaches a hardware implementation of its teachings, which includes its teachings of queues. <i>See, e.g.,</i> Van Loo ’485 at 7:50–53 (“The particular implementation of these logic modules [including the master transaction request output queues and system controller output queues], and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2.”); 10:30–32 (“The logic for the carrying out of the invention is provided by hardware/firmware logic of the SC ASIC and the master and slave interfaces, and by program instructions stored in memory 85 of the system, as shown in FIG. 1.”).

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Includes Logic (110, 112), MASTER QUEUE (290), Config Register (270), and Counter (310). Master 2 (125): Includes Logic (122), MASTER QUEUE (300), Config Register (280), and Counter (320). Slave 1 (140): Includes Logic (142), Slave Queue (185), ID Register (188), and Counter (190). Slave 2 (150): Includes Logic (152), Slave Queue (195), ID Register (190), and Counter (170). System Controller (130): Includes SC LOGIC (245), SCID Register (260), SCID Register (250), SCID Register (255), SCID Register (258), SCID Register (265), SCID Register (270), SCID Register (275), SCID Register (280), SCID Register (285), SCID Register (290), SCID Register (295), SCID Register (300), SCID Register (305), SCID Register (310), SCID Register (315), SCID Register (320), SCID Register (325), SCID Register (330), SCID Register (335), SCID Register (340), SCID Register (345), SCID Register (350), SCID Register (355), SCID Register (360), SCID Register (365), SCID Register (370), SCID Register (375), SCID Register (380), SCID Register (385), SCID Register (390), SCID Register (395), SCID Register (400), SCID Register (405), SCID Register (410), SCID Register (415), SCID Register (420), SCID Register (425), SCID Register (430), SCID Register (435), SCID Register (440), SCID Register (445), SCID Register (450), SCID Register (455), SCID Register (460), SCID Register (465), SCID Register (470), SCID Register (475), SCID Register (480), SCID Register (485), SCID Register (490), SCID Register (495), SCID Register (500), SCID Register (505), SCID Register (510), SCID Register (515), SCID Register (520), SCID Register (525), SCID Register (530), SCID Register (535), SCID Register (540), SCID Register (545), SCID Register (550), SCID Register (555), SCID Register (560), SCID Register (565), SCID Register (570), SCID Register (575), SCID Register (580), SCID Register (585), SCID Register (590), SCID Register (595), SCID Register (600), SCID Register (605), SCID Register (610), SCID Register (615), SCID Register (620), SCID Register (625), SCID Register (630), SCID Register (635), SCID Register (640), SCID Register (645), SCID Register (650), SCID Register (655), SCID Register (660), SCID Register (665), SCID Register (670), SCID Register (675), SCID Register (680), SCID Register (685), SCID Register (690), SCID Register (695), SCID Register (700), SCID Register (705), SCID Register (710), SCID Register (715), SCID Register (720), SCID Register (725), SCID Register (730), SCID Register (735), SCID Register (740), SCID Register (745), SCID Register (750), SCID Register (755), SCID Register (760), SCID Register (765), SCID Register (770), SCID Register (775), SCID Register (780), SCID Register (785), SCID Register (790), SCID Register (795), SCID Register (800), SCID Register (805), SCID Register (810), SCID Register (815), SCID Register (820), SCID Register (825), SCID Register (830), SCID Register (835), SCID Register (840), SCID Register (845), SCID Register (850), SCID Register (855), SCID Register (860), SCID Register (865), SCID Register (870), SCID Register (875), SCID Register (880), SCID Register (885), SCID Register (890), SCID Register (895), SCID Register (900), SCID Register (905), SCID Register (910), SCID Register (915), SCID Register (920), SCID Register (925), SCID Register (930), SCID Register (935), SCID Register (940), SCID Register (945), SCID Register (950), SCID Register (955), SCID Register (960), SCID Register (965), SCID Register (970), SCID Register (975), SCID Register (980), SCID Register (985), SCID Register (990), SCID Register (995). DataPath (155): Connects Master 1 and Master 2 to Slave 1 and Slave 2. DataPath Control (195): Connects Slave 1 and Slave 2 to the System Controller. I/O Devices & Memory (95): Connected to the System Controller. 100 FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a memory system architecture. On the left, multiple UPA ports (630, 640, ..., 640) are shown, each with a 'Dtag' block. These ports are connected to a central 'Datapath Crossbar NxM' (625). The crossbar is also connected to a 'System Controller & Address Crossbar' (650). The system controller is further connected to 'Memory Banks' (610, 620, ..., 620) and a 'Memory Databus' (690). A 'Memory Write Flow Control' block (600) is shown at the bottom, connected to the system controller and the memory banks. A 'Snoobus' is also depicted. Various control signals and data paths are labeled, including 'UPA_Databus', 'UPA_Addressbus', 'P_REQ', 'S_REPLY', 'P_REPLY', 'Addr_Valid', 'Data_Stall', 'ECC_Valid', 'Arb_Cntnl', 'Port_ID', 'Sys_Clk', 'Data', 'RAS/CAS', and 'Memory Control'. The diagram is labeled with reference numerals 600, 610, 620, 625, 630, 640, 650, 660, 670, 680, 690, 700, and 8. FIG. 5 Van Loo '485 at Figure 5.

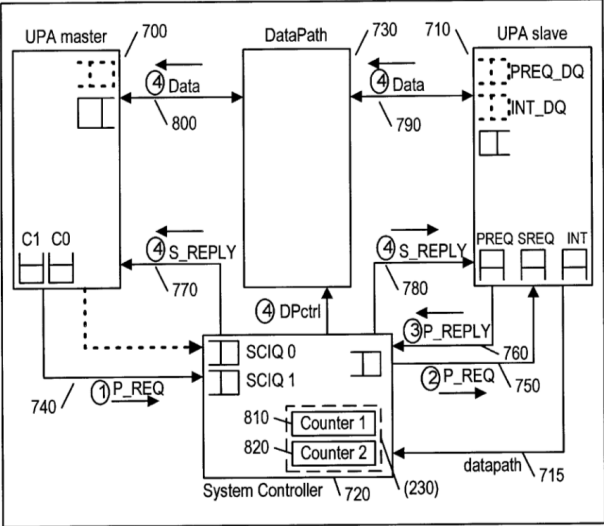
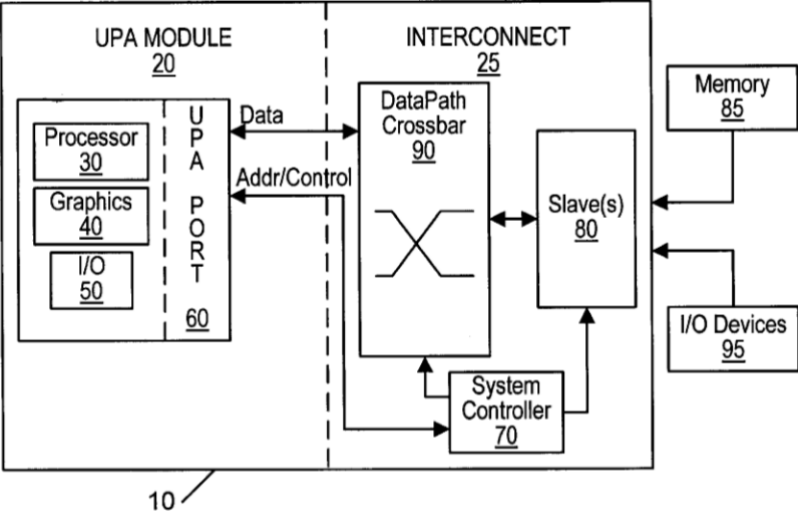
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “Masters 110 and 120 include data queues or buffers 115 and 125, and slaves 140 and 150 include data queues or buffers 185 and 195, whose functions are described in detail relative to FIGS. 6 and 7 below. The maximum sizes of the slave write data queues 185 and 195 are also represented by values in port ID registers 180 and 190, respectively. In the special case where there is a one-to-one correspondence to a request queue entry (e.g. 100) and a data buffer in the write data queue (e.g. 185), with the write data queue being maximally dimensioned to hold an entire packet (i.e. dimensioned such that it can hold the largest contemplated packet size), then the queue size in 180 can be represented by a single number.” Van Loo '485 at col. 6, ll. 36–48. “The masters 110 and 120 also include transaction request output queues 290 and 300, respectively, which are used to queue up transaction requests from each of the master interfaces to the SC 130. Each master 110 and 120 has a counter (310 and 320) used to track the number of pending transaction requests, as described below. The SC 130 is provided with output queues 210 and 220 and associated counters 230 and 240, respectively, whose operation will be described below.” Van Loo '485 at col. 7, ll. 12–20.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“Many combinations of standard hardware and software are possible to execute these functions in the system controller; and the same is true of the functions carried out in the slave devices (see slave logic modules 142 and 152) and the master devices (see master logic modules 112 and 122). Here, the logic modules represent all of the circuitry, programming, memory and intelligence necessary to carry out the functions of the invention as described; assembling the hardware and software to do so is a matter of routine to one skilled in the art, given the teaching of this invention. (Where a master device is a processor, the logic for implementing the present invention can of course be made up in large part of the processor itself and the instructions it executes.) The particular implementation of these logic modules, and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2.” Van Loo ’485 at col. 7, ll. 38–54. “Any master can request transactions to any slave via the SC, for any selected number of master and slaves. The SC will typically be an ASIC configured for a given system with a predetermined maximum number of master and slave interfaces. Since it is a relatively simple and inexpensive ASIC (by comparison with the system as a whole), it provides great flexibility and economy by allowing designers to easily configure different SC's at low cost for different systems, each one tailored to the specific needs of that system.” Van Loo ’485 at col. 10, ll. 21–30. “The logic for the carrying out of the invention is provided by hardware/firmware logic of the SC ASIC and the master and slave interfaces, and by program instructions stored in memory 85 of the system, as shown in FIG. 1. Alternative embodiments may implement the logic in other fashions, e.g. by providing memories and general purpose processors to carry out any of the steps executed by the master interfaces, system controller and slave interfaces of the preferred embodiment of this invention.” Van Loo ’485 at col. 10, ll. 31–39. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[1c] a first processing element having access to said input queue and said output queue;	Van Loo ’485 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches a first processing element having access to said input queue and said output queue, by disclosing, as non-limiting examples, a master or a master device. A master or a master device has access to both the input queue and output queue. In the second example theory, Van Loo ’485 teaches a first processing element having access to said input queue and said output queue, by disclosing, as non-limiting examples, a system controller. A system controller has access to both the input queue and output queue. <i>See, e.g.:</i> “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo ’485 at Abstract.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1 Van Loo '485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture centered around a DataPath Crossbar (92). On the left, three Master blocks (M1, M2, M3) are shown, each with the note "(can act as Slave)". They are connected to the crossbar via data paths D1, D2, and D3. A boot PROM (94) is also connected to the crossbar via data path D4. On the right, two Memory blocks (Mem1, Mem2) and two Slave blocks (S1, S2) are connected to the crossbar via data paths D4, D5, D6, and D7. A System Controller (75) is connected to the crossbar via a "DataPath Control" line and to each of the Master, boot PROM, and Slave blocks via address/control lines A/ctrl-1 through A/ctrl-7. The crossbar is represented by a box with an 'X' symbol and the number 92. FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Includes Logic (110, 112), MASTER QUEUE (290), Config Register (270), and Counter (310). Master 2 (125): Includes Logic (122), MASTER QUEUE (300), Config Register (280), and Counter (320). Slave 1 (140): Includes Logic (142), Slave Queue (185), ID Register (188), and Counter (190). Slave 2 (150): Includes Logic (152), Slave Queue (195), ID Register (198), and Counter (170). DataPath (155): A central component connected to Master 1, Master 2, Slave 1, and Slave 2 via Data lines. DataPath Control (160): Connected to the DataPath (155). System Controller (130): Includes SC LOGIC (245), SCID Register (265), SCID Register (255), SCID Register (250), SCID Register (252), Counter (240), Config Register (200), Counter (230), and Counter (210). I/O Devices & Memory (95): Connected to the System Controller (130). Arrows indicate data flow between components. A reference numeral 100 points to the overall system assembly. FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a memory system architecture. On the left, two UPA ports are shown: UPA port₀ (630) and UPA port₃₁ (640). Each port contains a Dtag block. UPA port₀ is connected to a UPA_Databus (632) and sends a P_REQ signal (1) to a Datapath Crossbar Nxm (625). The Datapath Crossbar Nxm (625) is connected to a UPA_Addressbus (626) and a Memory_Databus (690). The Memory_Databus (690) is connected to a series of Memory Banks (610, 620, ..., 610). A System Controller & Address Crossbar (650) is connected to the UPA_Addressbus (626) and the Memory_Databus (690). The System Controller & Address Crossbar (650) also controls a Coherence/Memory/Datapath Control block (660). The Coherence/Memory/Datapath Control block (660) is connected to the UPA ports (630, 640) and the Datapath Crossbar Nxm (625). A Memory Write Flow Control block (600) is connected to the UPA ports (630, 640) and the Coherence/Memory/Datapath Control block (660). A Snoobus is connected to the Dtag blocks (634, 644) and the Coherence/Memory/Datapath Control block (660). A Diag Control block is connected to the Coherence/Memory/Datapath Control block (660). A RAS/CAS block (670) is connected to the Memory Banks (610, 620, ..., 610) and the Memory_Databus (690). The diagram is labeled FIG. 5. FIG. 5 Van Loo '485 at Figure 5.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 – col. 3, l. 6. “The present invention uses a new system interconnect architecture and concomitant new methods for utilizing the interconnect to control transaction requests and data flow between master devices and slave or memory devices.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	In FIG. 1, the system 10 includes a UPA module 20 and an interconnect network or module 25, which in different embodiments of the invention may or may not be connected to the data path. The UPA module may include such devices as a processor 30, a graphics unit 40, and an I/O unit 50. Other units may be included, and act as the master units for the purposes of the present invention. A master interface is defined as the interface for any entity initiating transaction requests; examples of such masters are a CPU making memory requests, or an I/O channel and bridges making DMA requests.” Van Loo ’485 at col. 4, ll. 27–41. “In general, in this application a master is exemplified by a processor. However, a master may be any transaction-requesting device, whether or not it includes a microprocessor. Similarly, a "slave" refers herein to any device that can accept a transaction request, including both memory and non-memory devices, etc., and including devices such as processors and I/O controllers that may themselves act as masters.” Van Loo ’485 at col. 4, ll. 42–49. “In a preferred embodiment, the system controller 70 and UPA interface 60 are carried on the main processor chip, and the slave interface is on the motherboard, though many variations are possible. More generally, each master (whether a processor or some other device) has a UPA master interface, and each slave includes a UPA slave interface. The system controller in each case resides with the system.” Van Loo ’485 at col. 5, ll. 4–11. “A datapath crossbar 90 is also included in the interconnect module 25, and is coupled to the slave interface(s), the system controller 70, and the ports 60. The datapath crossbar may be a simple bus or may be a more complicated crossbar. (The UPA ports 60 may be configured as part of either the UPA module 20 or the interconnect module 25.) The datapath unit 90 is used to transmit read and write data in a manner to be described below.” Van Loo ’485 at col. 5, ll. 12–19. “FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31. “Each slave 140 and 150 includes a slave queue (160 and 170, respectively) for receiving transaction requests. The maximum sizes of these slave queues are represented by values in port ID registers 180 and 190, respectively.” Van Loo ’485 at col. 6, ll. 32–35. “Masters 110 and 120 include data queues or buffers 115 and 125, and slaves 140 and 150 include data queues or buffers 185 and 195, whose functions are described in detail relative to FIGS. 6 and 7 below. The maximum sizes of the slave write data queues 185 and 195 are also represented by values in port ID registers 180 and 190, respectively. In the special case where there is a one-to-one correspondence to a request queue entry (e.g. 100) and a data buffer in the write data queue (e.g. 185), with the write data queue being maximally dimensioned to hold an entire packet (i.e. dimensioned such that it can hold the largest contemplated packet size), then the queue size in 180 can be represented by a single number.” Van Loo ’485 at col. 6, ll. 36–48. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device;

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[1d] at least one second processing element in communication with said first processing element;	Van Loo ’485 teaches “at least one second processing element in communication with said first processing element,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches at least one second processing element in communication with said first processing element, by disclosing, as non-limiting examples, a system controller. A system controller, the second processing element is in communication with the first processing element. In the second example theory, Van Loo ’485 teaches at least one second processing element in communication with said first processing element, by disclosing, as non-limiting examples, a master or master device. A master or master device, the second processing element is in communication with the first processing element. <i>See, e.g.:</i> <pre> graph LR subgraph UPA_MODULE [UPA MODULE 20] P[Processor 30] G[Graphics 40] IO[I/O 50] UP[UPA PORT 60] P --- UP G --- UP IO --- UP end subgraph INTERCONNECT [INTERCONNECT 25] DPC[DataPath Crossbar 90] SC[System Controller 70] S[Slave(s) 80] DPC <--> SC DPC <--> S end UP <--> Data DPC UP <--> Addr/Control DPC S <--> M[Memory 85] S <--> ID[I/O Devices 95] </pre> FIG. 1 Van Loo ’485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture centered around a DataPath Crossbar (92). On the left, three Master blocks (M1, M2, M3) are shown, each with the note "(can act as Slave)". Master M1 is connected to the crossbar via data line D1 and address/control line A/ctrl-1. Master M2 is connected via D2 and A/ctrl-2. Master M3 is connected via D3 and A/ctrl-3. A boot PROM (94) is connected to the crossbar via D4 and A/ctrl-3. On the right, two Memory blocks (Mem1, Mem2) and two Slave blocks (S1, S2) are shown. Mem1 is connected via D4 and A/ctrl-7. Mem2 is connected via D5 and A/ctrl-6. S1 is connected via D6 and A/ctrl-5. S2 is connected via D7 and A/ctrl-4. A System Controller (75) is at the bottom, connected to the crossbar via a DataPath Control line and to the masters via A/ctrl-1, A/ctrl-2, and A/ctrl-3. The crossbar (92) is represented by a box with an 'X' symbol and a reference numeral 96. FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Contains Logic 110, MASTER QUEUE 112, Config Register 270, and Counter 310. It connects to the DataPath 155 via a Data bus. Master 2 (125): Contains Logic 122, MASTER QUEUE 123, Config Register 280, and Counter 320. It also connects to the DataPath 155 via a Data bus. Slave 1 (140): Contains Logic 142, Slave Queue 185, ID Register 188, and Counter 189. It connects to the DataPath 155 via a Data bus. Slave 2 (150): Contains Logic 152, Slave Queue 195, ID Register 190, and Counter 170. It connects to the DataPath 155 via a Data bus. DataPath 155: A central component that manages data flow between the masters and slaves. DataPath Control: A block that interfaces with the DataPath 155 and the System Controller. System Controller (130): Contains SC LOGIC 245, SCID Register 260, SCID Register 250, SCID Register 255, SCID Register 258, SCID Register 265, SCID Register 270, SCID Register 275, SCID Register 280, SCID Register 285, SCID Register 290, SCID Register 295, SCID Register 300, SCID Register 305, SCID Register 310, SCID Register 315, SCID Register 320, SCID Register 325, SCID Register 330, SCID Register 335, SCID Register 340, SCID Register 345, SCID Register 350, SCID Register 355, SCID Register 360, SCID Register 365, SCID Register 370, SCID Register 375, SCID Register 380, SCID Register 385, SCID Register 390, SCID Register 395, SCID Register 400, SCID Register 405, SCID Register 410, SCID Register 415, SCID Register 420, SCID Register 425, SCID Register 430, SCID Register 435, SCID Register 440, SCID Register 445, SCID Register 450, SCID Register 455, SCID Register 460, SCID Register 465, SCID Register 470, SCID Register 475, SCID Register 480, SCID Register 485, SCID Register 490, SCID Register 495, SCID Register 500, SCID Register 505, SCID Register 510, SCID Register 515, SCID Register 520, SCID Register 525, SCID Register 530, SCID Register 535, SCID Register 540, SCID Register 545, SCID Register 550, SCID Register 555, SCID Register 560, SCID Register 565, SCID Register 570, SCID Register 575, SCID Register 580, SCID Register 585, SCID Register 590, SCID Register 595, SCID Register 600, SCID Register 605, SCID Register 610, SCID Register 615, SCID Register 620, SCID Register 625, SCID Register 630, SCID Register 635, SCID Register 640, SCID Register 645, SCID Register 650, SCID Register 655, SCID Register 660, SCID Register 665, SCID Register 670, SCID Register 675, SCID Register 680, SCID Register 685, SCID Register 690, SCID Register 695, SCID Register 700, SCID Register 705, SCID Register 710, SCID Register 715, SCID Register 720, SCID Register 725, SCID Register 730, SCID Register 735, SCID Register 740, SCID Register 745, SCID Register 750, SCID Register 755, SCID Register 760, SCID Register 765, SCID Register 770, SCID Register 775, SCID Register 780, SCID Register 785, SCID Register 790, SCID Register 795, SCID Register 800, SCID Register 805, SCID Register 810, SCID Register 815, SCID Register 820, SCID Register 825, SCID Register 830, SCID Register 835, SCID Register 840, SCID Register 845, SCID Register 850, SCID Register 855, SCID Register 860, SCID Register 865, SCID Register 870, SCID Register 875, SCID Register 880, SCID Register 885, SCID Register 890, SCID Register 895, SCID Register 900, SCID Register 905, SCID Register 910, SCID Register 915, SCID Register 920, SCID Register 925, SCID Register 930, SCID Register 935, SCID Register 940, SCID Register 945, SCID Register 950, SCID Register 955, SCID Register 960, SCID Register 965, SCID Register 970, SCID Register 975, SCID Register 980, SCID Register 985, SCID Register 990, SCID Register 995. I/O Devices & Memory (95): Connected to the System Controller via a bidirectional bus. 100 FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 5 Van Loo '485 at Figure 5.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture for slave read flow control. It includes a UPA master (700) connected to a central DataPath (730), which is further connected to a UPA slave (710). The UPA master contains components C1 and C0. The UPA slave contains PREQ_DQ and INT_DQ signals. A System Controller (720) is shown at the bottom, containing SCIQ 0, SCIQ 1, Counter 1 (810), and Counter 2 (820). Signal paths are numbered: ① P_REQ from C0 to SCIQ 0; ② P_REQ from SCIQ 1 to the UPA slave's datapath (715); ③ P_REPLY from the UPA slave back to SCIQ 1; ④ S_REPLY from SCIQ 0 to C1; ⑤ DPctrl from the DataPath to SCIQ 0. Other signals include PREQ, SREQ, and INT between the UPA slave and the System Controller. FIG. 6 Van Loo '485 at Figure 6. "A UPA port 60 couples the module 20 to a system interconnect controller (SC) 70, which is in turn coupled to one or more slave interface(s) 80. The slave interface may be an interface for memory (such as main memory), an I/O interface, a graphics frame buffer, one or more bridges to other interconnection networks, or even a CPU receiving transactions to be serviced. In general, any device that accepts transaction requests for servicing may be given a slave interface 80 in accordance with the invention, such as conventional memory device(s) 85 and/or standard I/O device(s) 95." Van Loo '485 at col. 4, l. 60 – col. 5, l. 3. "A datapath crossbar 90 is also included in the interconnect module 25, and is coupled to the slave interface(s), the system controller 70, and the ports 60. The datapath crossbar may be s simple bus or may be a more complicated crossbar. (The UPA ports 60 may be configured as part of either the UPA module 20 or the interconnect module 25.) The datapath unit 90 is used to transmit read and write data in a manner to be described below." Van Loo '485 at col. 5, ll. 12–19. "FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31. “(It should be noted throughout the present description that for the sake of clarity split address and data buses are assumed and illustrated; however, the present invention is equally applicable to systems using shared address/data buses.) The I/O controller M3 accesses the boot PROM 94 to read the code for the initialization sequence, and sends it via line A/ctrl-3 to the SC 75, which sends it on to the processor M1.” Van Loo ’485 at col. 7, l. 67 – col. 8, l. 6. “When this complete-transaction signal is received by the master interface 110 (box 500), the master interface decrements the counter associated with that SCIQ (box 530) and proceeds to the step at box 440. At box 440, the counter 320 is incremented by one, and at box 450 the transaction request is sent by the master to the SC. Thus, the counter 320 now reflects the sending of one (or one additional) transaction request.” Van Loo ’485 at col. 11, ll. 39–46. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[1e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;	Van Loo ’485 teaches “a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element, by disclosing, as non-limiting examples, a counter on the system controller and/or a config register. A counter on the system controller and/or a config register stores a value of indicative space and is accessible by both the first and second processing elements discussed above. In the second example theory, Van Loo ’485 teaches a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element, by disclosing, as non-limiting examples, a master counter. A master counter stores a value of indicative space and is accessible by both the first and second processing elements discussed above. Defendants have proposed a construction that “register” means “set of flip–flops that store a set of related bits.” Van Loo ’485 teaches a reservation register that is a set of flip–flops that store a set of related bits. Van Loo ’485, 7:29–32 (“an ASIC may be provided for carrying out the transaction request handling, queue control, numerical comparisons and counting, etc.”); 7:50–53; 10:30–32. For example, the reservation registers identified above is explained and described as a register and the description of its behavior is consistent with a set of flip–flops that store a set of related bits. <i>See id.</i> Van Loo is a multiprocessor system

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	that uses registers. A POSITA reading Van Loo would have understood the description of these registers to be a set of flip-flops that store a set of related bits in view of the relevant disclosures, and further because that is how hardware systems and registers in particular had been designed and implemented for decades. <i>See, e.g.:</i> <pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- No --> 422[End 422] 420 -- Yes --> 424{For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424} 424 -- No --> 426[Wait 426] 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 (Fig. 3B)]) 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 440 500 -- No --> 510[Wait 510] 510 --> 440 </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction: SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- Yes --> CompleteMsg[Transaction complete message received from slave? 480] Decision4 -- No --> IncCounter[Increment SC counter for recipient slave by 1 482] IncCounter --> SendReq[Send transaction request to recipient slave 484] SendReq --> Note486[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 486] Note486 --> Decision4 CompleteMsg -- Yes --> DecCounter[Decrement SC counter for recipient slave 488] DecCounter --> Wait490[Wait 490] Wait490 --> Decision4 CompleteMsg -- No --> SendReq Note486 --> Decision4 Note486 --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 – col. 3, l. 6. “Initialization Operation The basic steps that take place at initialization include:

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(1) determine the sizes of the respective receive queues of all the slaves coupled to the system; (2) store the sizes of the slave receive queues in registers within the system controller; (3) determine the sizes of the system controller's receive queues; and (4) store the sizes of the system controller receive queues in predetermined registers in the master(s). Thus, at system initialization, the initialization software reads the contents of the size fields for the request queue and write data queue in each slave, and then copies these values into corresponding fields inside the configuration (config) register 200 of the SC 130. In one embodiment, the values in ID registers 170 and 180 (representing the slave queue sizes) are stored in separate fields in configuration ("config") register 200 of the SC 130. In addition, the values of the SCID registers 255 and 265 (representing the SC queue sizes) are stored in config registers 270 and 280, respectively, of the master interfaces 110 and 120." Van Loo '485 at col. 6, l. 53 – col. 7, l. 7. "Alternatively, config register 200 may be replaced by a separate configuration register for each UPA port implemented in the given SC. In this case, there would be two separate config registers, one for each of slaves 140 and 150." Van Loo '485 at col. 7, ll. 8–11. "Many combinations of standard hardware and software are possible to execute these functions in the system controller; and the same is true of the functions carried out in the slave devices (see slave logic modules 142 and 152) and the master devices (see master logic modules 112 and 122). Here, the logic modules represent all of the circuitry, programming, memory and intelligence necessary to carry out the functions of the invention as described; assembling the hardware and software to do so is a matter of routine to one skilled in the art, given the teaching of this invention. (Where a master device is a processor, the logic for implementing the present invention can of course be made up in large part of the processor itself and the instructions it executes.) The particular implementation of these logic modules, and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2." Van Loo '485 at col. 7, ll. 38–54. "Once the processor M3 has retrieved the initialization sequence instructions from the boot PROM 94, they are executed. The first operation is to read the BD registers of the memories and slaves. These ID registers, as described above with respect to FIG. 2, contain the values of the respective slaves' instruction queues and write data queues. The flow control sequence that is followed for this read operation follows that described

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	below for the Slave Read Flow Control in FIG. 6, the data from the ID registers being retrieved via a data bus (or datapath) 715. The ID register values are written to the config registers (such as config register 200) of the system controller (75 in FIG. 1A, 130 in FIG. 2). As discussed above, there is one config register per slave, or at least one field in a config register for each slave. The flow sequence followed for this write operation is as discussed below relative to FIG. 7. The I/O controller for the system is used for this purpose. Thus, assuming in FIG. 7 that for this operation the slave 710 is the I/O controller, the master (in this case, a processor) 700 causes the SC 720 to write the ID register values from each slave to its own config registers. In each case, the respective ID register value is stored in a buffer of the processor (master 700 in FIG. 7 or master M1 in FIG. 1A), and this value is passed to the system controller to the I/O controller (slave 710 in FIG. 7 or master/slave M3 in FIG. 1A), which then writes it right back to the system controller via the datapath provided for that purpose (data bus 715 in FIG. 7). The next step in the initialization procedure is to read the sizes of the receive queues of the system controller (e.g. the SCIQ's 0 and 1 shown in FIG. 7 or SCIQ's 250 and 260 in FIG. 2). The receive queue sizes are stored in the SCID registers (see registers 255 and 265 shown in FIG. 2). This read operation is executed using the I/O controller of the system, resulting in the master/processor storing the SC receive queue values in a buffer or preassigned register. Finally, these SCIQ sizes are written into the master config registers (such as 270 and 280 in FIG. 2). If the system is a uniprocessor system, then this amounts the processor writing the SCID values to its own config register and to the config registers of other devices that can act as masters. If it is a multiprocessor system, then one processor acts as a master and writes SCID values to both its own config register and to those of the other processors." Van Loo '485 at col. 8, ll. 12–54. "After initialization of the config register 200 in the SC 130 and the config registers 270 and 280 in the masters, normal operation of the system 100 can commence. During operation, the SC maintains in its config register 200 a copy of the respective values of the slave ID registers 180 and 190, and hence "knows" the maximum number of transaction requests that each slave interface can handle in its slave request queue (160 or 170), and the maximum amount of data that can be held in its slave data queue (185 or 195). At any given time, the counters 230 and 240 store the number of pending transaction requests in the corresponding slave request queue, and the size of the pending store data in the slave store data queue. Unissued transaction requests may in some circumstances be stored for the slaves 140 and 150 in output queues 210 and 220, which may be arbitrarily large, and in particular may be larger than the SCIQ's 250 and 260. In other

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	circumstances, requests remain enqueued in corresponding SCIQ's." Van Loo '485 at col. 8, l. 61 – col. 9, l. 11. "Referring now to FIGS. 3A-3B, at initialization (box/method step 400) all UPA port ID registers (e.g. the slave ID registers 180 and 190 in FIG. 2) are read, and their contents are written into the appropriate fields in the SC config register 200 (or into separate, dedicated config registers, as discussed above). The separate fields in a single SC config register embodiment is more likely to be used when the UPA (slave) ports are configured with a PROM instead of a port ID register. In the present application, whenever fields of the config register are referred to, it may be taken alternatively to mean separate config registers, and vice versa. At box 410, the master registers are now initialized, which involves reading the SCID registers 255 and 265 and writing the SCIQ sizes (stored in those registers) in the respective config registers 270 and 280." Van Loo '485 at col. 10, ll. 41–55. "When either the read or the write operation is ready for execution as far as the master interface is concerned, then at box 430 the system tests whether the value of the master counter (in this example, counter 320) or equal to the value stored in the config register, i.e. the size of the associated SCIQ 260 (as originally provided by the SCID register 265). (The master counter should never be able to exceed the value stored in the config registers, but in case it did this could be taken into account by using a "$\geq$" instead of "=" in the comparison test.) If the counter has not issued requests equal to the total SCIQ 260 size, then this test will be false and the method proceeds to box 440." Van Loo '485 at col. 11, ll. 18–29. "a first register storing a first register value representing the number of locations in said slave receive queue available for queueing of transaction requests by said least one slave device; a first counter storing a first counter value representing at any given time the number of transaction requests that are stored in said slave receive queue;" Van Loo '485 at col. 16, ll. 46–52. "a first register storing a first register value representing the number of locations in said slave receive queue available for queueing of transaction requests by said least one slave device; a first counter storing a first counter value representing at any given time the number of transaction requests that are stored in said slave receive queue;" Van Loo '485 at col. 17, ll. 15–21.
[1f] computer storage storing instructions, which when executed	Van Loo '485 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
by said at least one second processing element:	Van Loo '485 supports at least two example theories for invalidity. In the first example theory, Van Loo '485 teaches computer storage storing instructions that are executed by said at least one second processing element, by disclosing, as non-limiting examples, ROM or RAM storing program instructions to execute system controller functions or boot PROM storing the initialization sequence. The second processing element (e.g., a system controller) executes instructions those instructions stored in the computer storage. In the second example theory, Van Loo '485 teaches computer storage storing instructions that are executed by said at least one second processing element, by disclosing, as non-limiting examples, system storing program instructions to execute master interface functions or boot PROM storing the initialization sequence. The second processing element (e.g., a master or a master device) executes instructions those instructions stored in the computer storage. <i>See, e.g.:</i> <pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- No --> 422[End 422] 420 -- Yes --> 424 424 --> 426{Wait 426} 426 --> 424 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454((to box 454 Fig. 3B)) 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 500 -- No --> 510[Wait 510] 510 --> 520[Decrement Master Counter by 1 520] 520 --> 440 </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision554{Is this a memory operation or a slave operation? 554} Decision554 -- "(Memory operation)" --> Decision456{Is the recipient memory valid, given ordering constraints? 456} Decision456 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision456 Decision456 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] Decision554 -- "(Slave operation)" --> Determine462[Determine recipient slave for transaction request 462] Determine462 --> Decision464{is the recipient slave valid, given ordering constraints? 464} Decision464 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision464 Decision464 -- Yes --> Move468[Move transaction onto SCOQ for the recipient slave 468] Move468 --> Note470[If this is a slave write transaction: SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision475{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 475} Decision475 -- No --> Increment480[Increment SC counter for recipient slave by 1 480] Decision475 -- Yes --> Send485[Send transaction request to recipient slave 485] Increment480 --> Send485 Send485 --> Note495[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 495] Note495 --> Decision530{Transaction complete message received from slave? 530} Decision530 -- No --> Wait540[Wait 540] Wait540 --> Decision530 Decision530 -- Yes --> Decrement550[Decrement SC counter for recipient slave 550] Decrement550 --> Decision554</pre> FIG. 3B Van Loo '485 at Figure 3B.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “One or more conventional memories or other data storage devices 85 and one or more input/output (I/O) devices 95 forming part of the system 10 are provided for user interface, data output, etc.; these various slave devices may include RAM, ROM, disk drives, monitors, keyboards, track balls, printers, etc. They are coupled into the interconnect module 25 via the slave interfaces 80. The "slave" designation means in this case only that such devices accept requests from one or more processors and fulfill those requests.” Van Loo '485 at col. 5, ll. 20–28. “The circuitry for carrying out the operations of the SC is indicated by SC logic module 245 in FIG. 2, and may include conventional hardwired and/or software logic for carrying out the necessary functions. For instance, an ASIC may be provided for carrying out the transaction request handling, queue control, numerical comparisons and counting, etc. that are used in the invention. Alternatively, a general purpose processor could be used, configured (such as by program instructions stored in an associated conventional memory, e.g. ROM or RAM) to execute the functions discussed herein.” Van Loo '485 at col. 7, ll. 27–37. “Many combinations of standard hardware and software are possible to execute these functions in the system controller; and the same is true of the functions carried out in the slave devices (see slave logic modules 142

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	and 152) and the master devices (see master logic modules 112 and 122). Here, the logic modules represent all of the circuitry, programming, memory and intelligence necessary to carry out the functions of the invention as described; assembling the hardware and software to do so is a matter of routine to one skilled in the art, given the teaching of this invention. (Where a master device is a processor, the logic for implementing the present invention can of course be made up in large part of the processor itself and the instructions it executes.) The particular implementation of these logic modules, and the extent to which it is represented by software or hardware, are widely variable and thus shown only in block form in FIG. 2.” Van Loo ’485 at col. 7, ll. 38–54. “The logic for the carrying out of the invention is provided by hardware/firmware logic of the SC ASIC and the master and slave interfaces, and by program instructions stored in memory 85 of the system, as shown in FIG. 1. Alternative embodiments may implement the logic in other fashions, e.g. by providing memories and general purpose processors to carry out any of the steps executed by the master interfaces, system controller and slave interfaces of the preferred embodiment of this invention.” Van Loo ’485 at col. 10, ll. 31–39. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and adapted for conditionally transferring said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value, wherein said first circuit is a circuit separate from said at least one microprocessor.” Van Loo ’485 at col. 16, ll. 55–63. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and configured to conditionally transfer said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value.” Van Loo ’485 at col. 17, ll. 25–32.
[1g] accesses said reservation register and reads said stored value;	Van Loo ’485 teaches “accesses said reservation register and reads said stored value,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches that the second processing element executes instructions that causes it to access said reservation register and reads said stored value, by disclosing, as non-limiting examples, the second processing element (e.g., a system controller) checking the reservation register (e.g., a

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	config register and/or a counter). The second processing element accesses and reads the stored value in the reservation register. In the second example theory, Van Loo '485 teaches that the second processing element executes instructions that causes it to access said reservation register and reads said stored value, by disclosing, as non-limiting examples, the second processing element (e.g., a master) checking the reservation register (e.g., a master counter). The second processing element accesses and reads the stored value in the reservation register. <i>See, e.g.:</i> “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo '485 at Abstract.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420[New Transaction request 420] 495([from box 495 (Fig. 3B)]) --> 420 420 --> 424[For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424] 424 -- No --> 426[Wait 426] 426 --> 424 424 -- Yes --> 430[Is master counter value = total SCIQ size? 430] 430 -- Yes --> 500[Transaction complete message (S_REPLY) received from SC? 500] 500 -- No --> 510[Wait 510] 510 --> 500 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 440[Send transaction request to SC and insert at head of SCIQ 440] 430 -- No --> 440 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 (Fig. 3B)]) </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction, SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- Yes --> CompleteMsg[Transaction complete message received from slave? 480] Decision4 -- No --> IncSC[Increment SC counter for recipient slave by 1 482] IncSC --> SendReq[Send transaction request to recipient slave 484] SendReq --> Note486[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 486] Note486 --> Decision4 CompleteMsg -- Yes --> DecSC[Decrement SC counter for recipient slave 490] DecSC --> Wait492[Wait 492] Wait492 --> Decision4 CompleteMsg -- No --> SendReq</pre> FIG. 3B Van Loo '485 at Figure 3B.

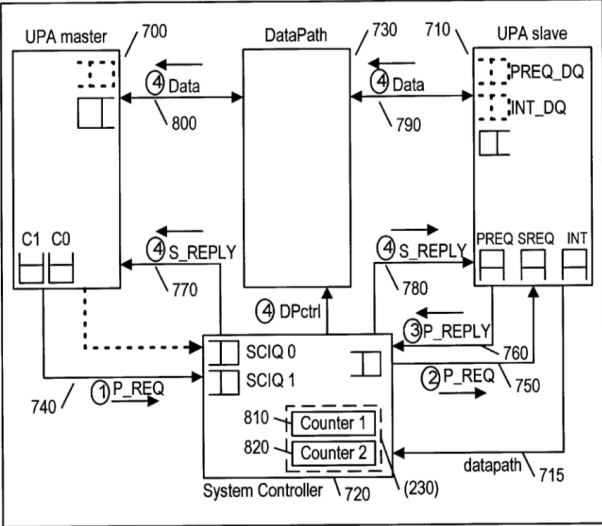
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The SC 130 then determines to which of the two slaves 140 and 150 the transaction request is destined, and checks the counter for that queue. For instance, if slave 140 is the destination for the transaction request, then the SC 130 checks the counter 210 and compares the value stored there with the value in the config register 200 corresponding to the ID register 180. If the counter 230 value is less than the value stored in the config register, then the SC 130 issues the transaction request and increments the counter 230. Otherwise, the transaction request is maintained in the output queue 210. (In some transactions related to ordering constraints for transactions for different requests from the same master, it may be desirable to leave the request in the SCIQ 250.)” Van Loo '485 at col. 9, ll. 19–32. “At box 424, if the pending transaction request is for a read operation, then the system determines whether the master read data buffer (discussed in greater detail below) for the master interface is ready to accept data, i.e. whether there is sufficient room in the master read data buffer to receive the data to be read. If not, then the system waits as at box 426 until the master read data buffer is ready. Note that a write operation need not be held up during a wait period for a read operation, but may proceed independently; and vice versa.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	For a write operation, the system determines whether the data to be written to one of the slaves via a slave interface or memory is in fact ready for writing in (transmission to) a master write buffer. If not, again at box 426 a wait period is executed, until the data item is ready for writing.” Van Loo ’485 at col. 11, ll. 4–17. “When either the read or the write operation is ready for execution as far as the master interface is concerned, then at box 430 the system tests whether the value of the master counter (in this example, counter 320) or equal to the value stored in the config register, i.e. the size of the associated SCIQ 260 (as originally provided by the SCID register 265). (The master counter should never be able to exceed the value stored in the config registers, but in case it did this could be taken into account by using a “$\geq$” instead of “=” in the comparison test.) If the counter has not issued requests equal to the total SCIQ 260 size, then this test will be false and the method proceeds to box 440.” Van Loo ’485 at col. 11, ll. 18–29. “At box 475, the SC 130 then checks the counter for the recipient slave, e.g. counter 240 if slave 150 is the destination for the pending transaction request. If the counter equals or exceeds the value in the config register (i.e. the size indicated by the ID register 180 or 190, which were read at initialization), then the request is not yet allowed. In this case, then steps 530-550 are followed (essentially identical to steps 500-520), until a free line opens up in the destination slave queue to receive the transaction request. If the appropriate counter (230 or 240) has not reached its maximum allowed value, then it is incremented by one (box 480), and the transaction request is sent to the recipient slave (box 490).” Van Loo ’485 at col. 12, ll. 50–62. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and adapted for conditionally transferring said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value, wherein said first circuit is a circuit separate from said at least one microprocessor.” Van Loo ’485 at col. 16, ll. 55–63. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and configured to conditionally transfer said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value.” Van Loo ’485 at col. 17, ll. 25–32. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60. “9. The method of claim 6, further including the steps of: storing a pending request value representing the total number of transactions that said recipient slave request queue can receive; and maintaining a count of the number of transaction requests present in the recipient slave request queue; wherein step 4 includes the step of comparing said count with said pending request value, to determine that the recipient slave request queue can accept the transaction request if said count is less than said pending request value.” Van Loo ’485 at col. 18, ll. 1–11.
[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;	Van Loo ’485 teaches “determines when said read value indicates available space in said input queue for said first processing element to issue a command,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches that the second processing element executes instructions that causes it to determine when said read value indicates available space in said input queue for said first processing element to issue a command, by disclosing, as non-limiting examples, the second processing

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	element (e.g., a system controller) determining if there is enough space for a transaction request by comparing a counter value to a config register value. The second processing element determines if the stored value indicates available space in the input queue for the first processing element to issue a command. In the second example theory, Van Loo '485 teaches that the second processing element executes instructions that causes it to determine when said read value indicates available space in said input queue for said first processing element to issue a command, by disclosing, as non-limiting examples, the second processing element (e.g., a master device) determining if there is enough space for a transaction request by checking a master counter. The second processing element determines if the stored value indicates available space in the input queue for the first processing element to issue a command. “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo '485 at Abstract.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- from box 495 (Fig. 3B) --> 420 420 -- No --> 422[End 422] 420 -- Yes --> 424{For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424} 424 -- No --> 426[Wait 426] 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450[Increment master counter by 1 450] 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 500 -- No --> 510[Wait 510] 510 --> 520[Decrement Master Counter by 1 520] 520 --> 452{If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452} 450 --> 452 452 --> 454([to box 454 (Fig. 3B)]) </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction: SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- Yes --> CompleteMsg[Transaction complete message received from slave? 530] CompleteMsg -- No --> Wait540[Wait 540] Wait540 --> CompleteMsg CompleteMsg -- Yes --> DecSC[Decrement SC counter for recipient slave 550] DecSC --> DecSC DecSC --> IncSC[Increment SC counter for recipient slave by 1 480] IncSC --> SendReq[Send transaction request to recipient slave 482] SendReq --> Note496[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 496] Note496 --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

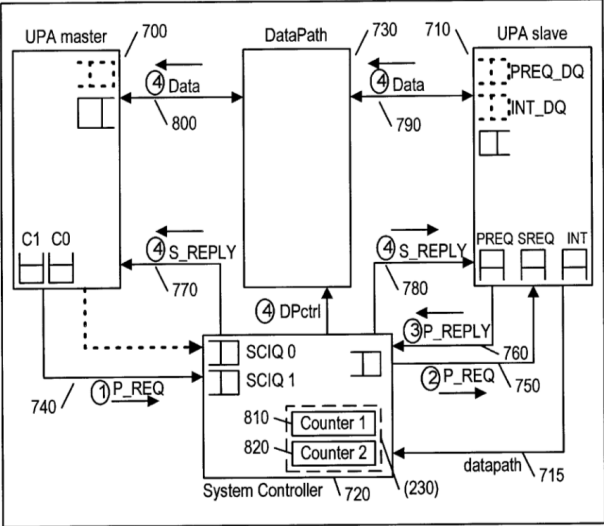
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The SC 130 then determines to which of the two slaves 140 and 150 the transaction request is destined, and checks the counter for that queue. For instance, if slave 140 is the destination for the transaction request, then the SC 130 checks the counter 210 and compares the value stored there with the value in the config register 200 corresponding to the ID register 180. If the counter 230 value is less than the value stored in the config register, then the SC 130 issues the transaction request and increments the counter 230. Otherwise, the transaction request is maintained in the output queue 210. (In some transactions related to ordering constraints for transactions for different requests from the same master, it may be desirable to leave the request in the SCIQ 250.)” Van Loo '485 at col. 9, ll. 19–32. “At box 424, if the pending transaction request is for a read operation, then the system determines whether the master read data buffer (discussed in greater detail below) for the master interface is ready to accept data, i.e. whether there is sufficient room in the master read data buffer to receive the data to be read. If not, then the system waits as at box 426 until the master read data buffer is ready. Note that a write operation need not be held up during a wait period for a read operation, but may proceed independently; and vice versa.

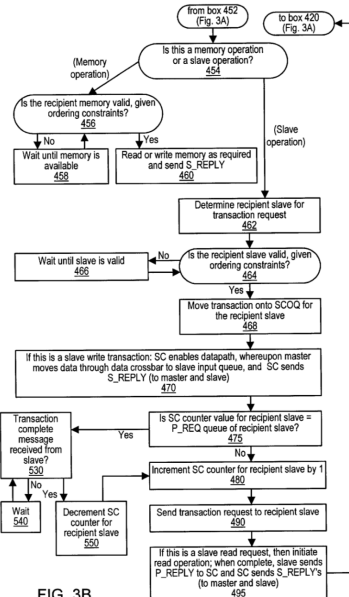
Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	For a write operation, the system determines whether the data to be written to one of the slaves via a slave interface or memory is in fact ready for writing in (transmission to) a master write buffer. If not, again at box 426 a wait period is executed, until the data item is ready for writing.” Van Loo ’485 at col. 11, ll. 4–17. “When either the read or the write operation is ready for execution as far as the master interface is concerned, then at box 430 the system tests whether the value of the master counter (in this example, counter 320) or equal to the value stored in the config register, i.e. the size of the associated SCIQ 260 (as originally provided by the SCID register 265). (The master counter should never be able to exceed the value stored in the config registers, but in case it did this could be taken into account by using a “$\geq$” instead of “=” in the comparison test.) If the counter has not issued requests equal to the total SCIQ 260 size, then this test will be false and the method proceeds to box 440.” Van Loo ’485 at col. 11, ll. 18–29. “At box 475, the SC 130 then checks the counter for the recipient slave, e.g. counter 240 if slave 150 is the destination for the pending transaction request. If the counter equals or exceeds the value in the config register (i.e. the size indicated by the ID register 180 or 190, which were read at initialization), then the request is not yet allowed. In this case, then steps 530-550 are followed (essentially identical to steps 500-520), until a free line opens up in the destination slave queue to receive the transaction request. If the appropriate counter (230 or 240) has not reached its maximum allowed value, then it is incremented by one (box 480), and the transaction request is sent to the recipient slave (box 490).” Van Loo ’485 at col. 12, ll. 50–62. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and adapted for conditionally transferring said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value, wherein said first circuit is a circuit separate from said at least one microprocessor.” Van Loo ’485 at col. 16, ll. 55–63. “a first circuit coupled to said first register, first counter and slave receive queue, including comparison logic to compare said first register value with said first counter value, and configured to conditionally transfer said transaction requests from said interconnect receive queue to said slave device in dependence upon a difference between said first counter value and said first register value.” Van Loo ’485 at col. 17, ll. 25–32. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60. “9. The method of claim 6, further including the steps of: storing a pending request value representing the total number of transactions that said recipient slave request queue can receive; and maintaining a count of the number of transaction requests present in the recipient slave request queue; wherein step 4 includes the step of comparing said count with said pending request value, to determine that the recipient slave request queue can accept the transaction request if said count is less than said pending request value.” Van Loo ’485 at col. 18, ll. 1–11.
[1i] notifies said first processing element to issue said command to said input queue; and	Van Loo ’485 teaches “notifies said first processing element to issue said command to said input queue,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches that the second processing element executes instructions that causes it to notify said first processing element to issue said command to said input queue, by disclosing, as non-limiting examples, the second processing element (e.g., a system controller) sending notice to the first

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	processing element (e.g., a master device) that indicates transaction completion and notifies the first processing element that one more slot is available in the downstream queue, thereby enabling the first processing element to issue the next transaction request. The second processing element notifies the first processing element to issue the command to the input queue. In the second example theory, Van Loo '485 teaches that the second processing element executes instructions that causes it to notify said first processing element to issue said command to said input queue, by disclosing, as non-limiting examples, the second processing element (e.g., a master) sending notice to the first processing element (e.g., a system) that indicates transaction completion and notifies the first processing element that one more slot is available in the downstream queue, thereby enabling the first processing element to issue the next transaction request. The second processing element notifies the first processing element to issue the command to the input queue. <i>See, e.g.:</i> <pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- No --> 422[End 422] 420 -- Yes --> 424{For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424} 424 -- No --> 426[Wait 426] 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 500 --> 510[Wait 510] 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 (Fig. 3B)]) 422 --> 495([from box 495 (Fig. 3B)]) </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction, SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- Yes --> CompleteMsg[Transaction complete message received from slave? 480] Decision4 -- No --> IncCounter[Increment SC counter for recipient slave by 1 482] IncCounter --> SendReq[Send transaction request to recipient slave 484] SendReq --> Note486[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 486] Note486 --> CompleteMsg CompleteMsg -- No --> Wait490[Wait 490] CompleteMsg -- Yes --> DecCounter[Decrement SC counter for recipient slave 492] DecCounter --> Decision4 Wait490 --> Decision4 Note486 --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

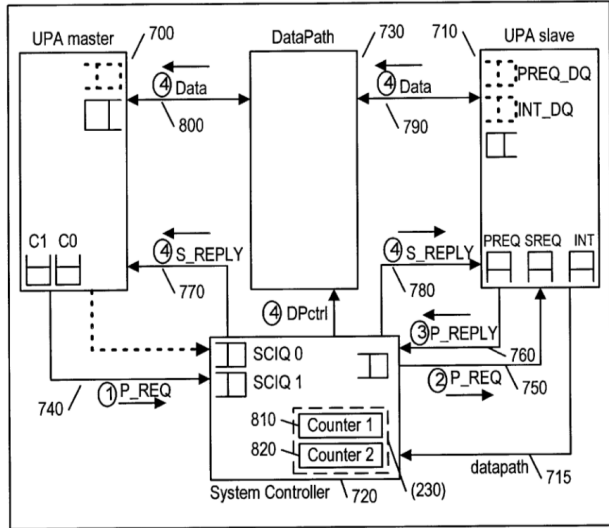
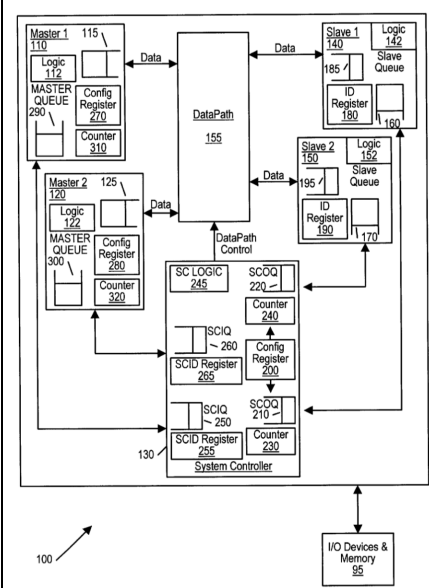
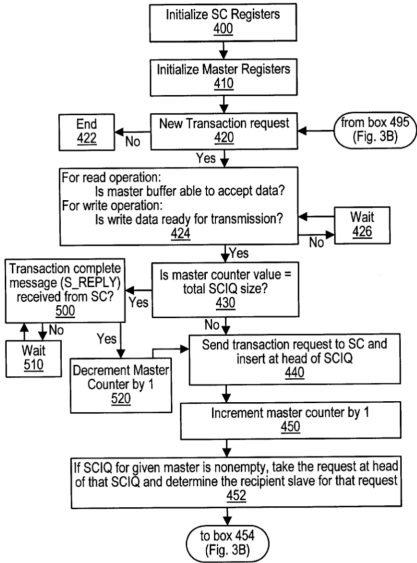
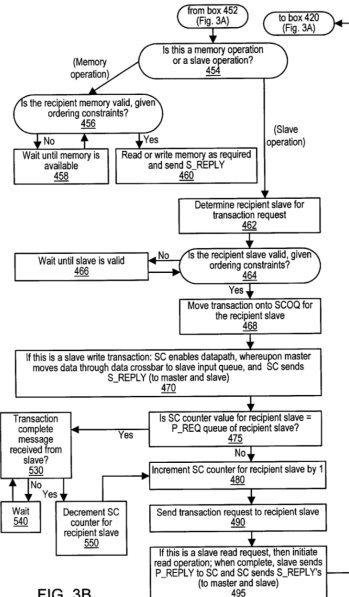
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The masters and the controller are prevented from issuing any transaction requests (or to initiate a write-data transfer requests) downstream when the respective counter indicates that the corresponding request or data queue downstream is full. When a transaction is complete (e.g. upon the receipt of requested data read or consumption of write data by the slave), the relevant counter is decremented to indicate the availability of a place in the transaction queue or write-data queue.” Van Loo '485 at col. 3, ll. 7–15. “Assuming the transaction request is issued in this example, then the SC 130 sends a signal to the master 110 to this effect (upon completion of the transaction, e.g. the transfer of data) and removes the transaction request from its input queue 250 (upon sending of the reply). The master 110 accordingly decrements its counter 310, which enables it to issue an additional pending transaction request. If the counter 310 was at its maximum (indicating that the SCIQ 250 was full), the decrementation of the counter 310 allows room for a single additional transaction request from the master 110 to the SC 130. If the counter 310 was not at its maximum value, then the decrementation of the counter 310 simply adds one to the number of transaction requests available to the master interface 110.” Van Loo '485 at col. 9, ll. 33–46.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“If the counter value has reached its maximum allowable value, then the transaction request will not be passed on to the SC, and the method proceeds to box 500. In this case, the transaction request pending in the master interface is required to wait (box 510) until a complete-transaction signal has been received from the SC before it can be issued. In a preferred embodiment, this complete-transaction signal takes the form of an S_REPLY signal, discussed in detail below with respect to FIGS. 4-7.” Van Loo ’485 at col. 11, ll. 30–38. “When this complete-transaction signal is received by the master interface 110 (box 500), the master interface decrements the counter associated with that SCIQ (box 530) and proceeds to the step at box 440. At box 440, the counter 320 is incremented by one, and at box 450 the transaction request is sent by the master to the SC. Thus, the counter 320 now reflects the sending of one (or one additional) transaction request.” Van Loo ’485 at col. 11, ll. 39–46.
[1j] wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue.	Van Loo ’485 teaches “wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue, by disclosing, as non-limiting examples, the first processing element (e.g., a master device) receiving the notification that the input queue has become available (e.g., a transaction-completion signal), issuing a transaction request to the input queue, and receiving a transaction-completion signal or a S_REPLY. The first processing element receives the notification, issues the command, and receives a corresponding response from the output queue. In the second example theory, Van Loo ’485 teaches wherein said first processing element receives notification from said at least one second processing element regarding issuing the command, issues said command to said input queue, and receives a response corresponding to said command from said output queue, by disclosing, as non-limiting examples, the first processing element (e.g., a system controller) receiving the notification that the input queue has become available (e.g., via receipt of a transaction request), issuing a transaction request to the input queue, and receiving a transaction-completion signal or a S_REPLY.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The first processing element receives the notification, issues the command, and receives a corresponding response from the output queue. <i>See, e.g.:</i>  The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Includes Logic (110), MASTER QUEUE (112), Config Register (270), and Counter (310). Master 2 (125): Includes Logic (122), MASTER QUEUE (300), Config Register (280), and Counter (320). Slave 1 (140): Includes Logic (142), Slave Queue (185), and ID Register (190). Slave 2 (150): Includes Logic (152), Slave Queue (195), and ID Register (170). System Controller (130): Includes SC LOGIC (245), SCIO (250), SCID Register (255), SCIOQ (210), SCIOQ Counter (230), SCIOQ Config Register (200), SCIOQ Counter (240), and SCIOQ Config Register (260). DataPath (155): A central data path connecting the masters and slaves. DataPath Control: A control unit for the data path. I/O Devices & Memory (85): Connected to the System Controller. Arrows indicate data flow: Master 1 and Master 2 send data to the DataPath, which then sends data to Slave 1 and Slave 2. The System Controller is connected to the DataPath and I/O Devices & Memory. FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- No --> 422[End 422] 420 -- Yes --> 424{For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424} 424 -- No --> 426[Wait 426] 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 Fig. 3B]) 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 500 -- No --> 510[Wait 510] 510 --> 420 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 420 </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction, SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- No --> IncrementSC[Increment SC counter for recipient slave by 1 480] Decision4 -- Yes --> SendRequest[Send transaction request to recipient slave 482] SendRequest --> Note496[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 496] Note496 --> Decision5{Transaction complete message received from slave? 530} Decision5 -- No --> Wait540[Wait 540] Wait540 --> Decision5 Decision5 -- Yes --> DecrementSC[Decrement SC counter for recipient slave 550] DecrementSC --> Decision4 Note496 --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

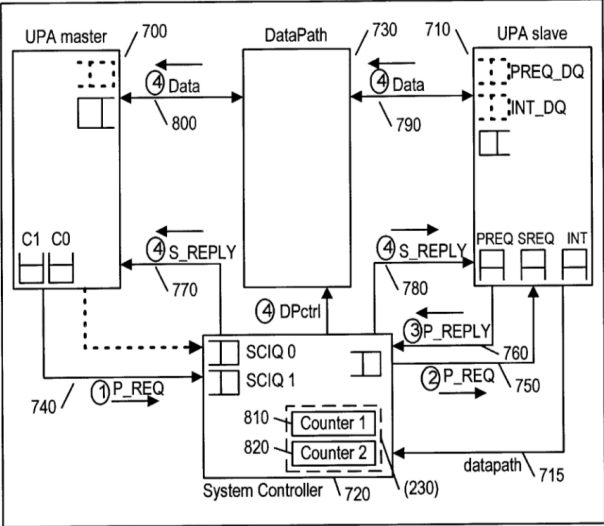
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “For the purposes of this invention, a "transaction" may be defined as a request packet issued by a master, followed by an acknowledgment packet (not necessarily a full packet, depending upon the chosen implementation) from the recipient immediately downstream. There may or may not be a data transfer accompanying a request packet, and the data transfer may either occur on the same set of wires as the request packet, or on separate datapath wires. This is described in greater detail below in connection with FIGS. 4-7.” Van Loo '485 at col. 4, ll. 50–59. “Assuming the transaction request is issued in this example, then the SC 130 sends a signal to the master 110 to this effect (upon completion of the transaction, e.g. the transfer of data) and removes the transaction request from its input queue 250 (upon sending of the reply). The master 110 accordingly decrements its counter 310, which enables it to issue an additional pending transaction request. If the counter 310 was at its maximum (indicating that the SC 250 was full), the decrementation of the counter 310 allows room for a single additional transaction request from the master 110 to the SC 130. If the counter 310 was not at its maximum value, then the decrementation of the counter 310 simply adds one to the number of transaction requests available to the master interface 110.” Van Loo '485 at col. 9, ll. 33–46.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“At box 420, it is determined whether a new transaction request is pending in one of the masters, e.g. the master 120.” Van Loo ’485 at col. 10, ll. 67–68. “When this complete-transaction signal is received by the master interface 110 (box 500), the master interface decrements the counter associated with that SCIQ (box 530) and proceeds to the step at box 440. At box 440, the counter 320 is incremented by one, and at box 450 the transaction request is sent by the master to the SC. Thus, the counter 320 now reflects the sending of one (or one additional) transaction request.” Van Loo ’485 at col. 11, ll. 39–46. “FIG. 6: read operation from a device other than memory;” Van Loo ’485 at col. 13, l. 21. “Memory Read Requests: FIG. 4 This read request example assumes that data is coming from memory, and not, e.g., from a cache. Snoop operations on the snoopbus shown in FIG. 4 are not in consideration here. Event 1: When a UPA master port such as post 630 has a read-from-memory transaction ready, and the master counter is not at its allowed maximum (see box 430 in FIGS. 3A-3B), the read transaction is issued on the UPA.. Addressbus from UPA port 630 to the system controller 650. This is indicate by the event 1 (P.. REQ) along the UPA.. Addressbus 660 in FIG. 4, with the direction of the information indicated by the arrow, i.e. from the port to the SC. Event 2: The memory cycle i.e. RAS (read-address-strobe)/CAS (column-address-strobe) request issuance! is issued over memory control bus 670 to the memory banks 610 . . . 620. See vent 2 ("RAS/CAS") along bus 670. Event 3: The datapath is scheduled by a signal along the datapath control bus 680, and data items are accordingly delivered from memory to the datapath crossbar 625 via a memory databus 690 and UPA databus 700. This fulfills the read request.” Van Loo ’485 at col. 14, ll. 4–25. “Slave Read Requests: FIG. 6 FIG. 6 illustrates a read sequence to a slave device other than memory, and is similar to FIG. 2, but for this example a single master interface 710 and a single slave interface 720 are used, coupled by a system controller 720 and a datapath crossbar 730. Event 1 indicates the issuance of a read request P.. REQ on UPA address bus 740 to SC 720.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	In event 2, the SC 720 sends the P.. REQ on bus 750 to the slave interface 710. To do this, if there are several slave interfaces, the SC must first decode the address to ensure that the P.. REQ goes to the correct slave interface. Event 2 informs the slave interface to prepare the data to move through the datapath. When the data items are ready, then event 3 takes place, namely the sending of a P.. REPLY from the slave 710 to the SC 720 over bus 760. In event 4, a series of steps are executed to cause the master interface to receive the data: SC 720 schedules the datapath 730, and issues an S.. REPLY over bus 770 to the master interface 700. In addition, the SC issues the S.. REPLY over bus 780 to the slave 710, to drive the data, when it is ready, on the slave's UPA databus 790 via the datapath and over the databus 800 to the master interface 700.” Van Loo ’485 at col. 14, l. 49 – col. 15, l. 6.
2. The system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer.	Van Loo ’485 teaches “[t]he system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer,” including at least the following exemplary citations. For example, Van Loo ’485 teaches wherein said input queue is a first-in-first-out (FIFO) buffer, by disclosing, as non-limiting examples, a slave device executes transaction requests in the order the transactions requests are received at the input queue. <i>See, e.g.:</i>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The SC is provided with intelligence and/or logic (hardware and/or software) to determine whether it has a transaction request pending in its request receive queue (such as SCIQ's 250 and 260). If so, then at box 452 the transaction request at the head of the queue is examined to determine which slave is intended as the recipient for the request. This queue control and recipient determination is carried out in a conventional manner.” Van Loo '485 at col. 11, ll. 55–62. “As a rule, in this operation the order of reception of the transaction requests will be the order of reply by the slave interfaces.” Van Loo '485 at col. 13, ll. 55–57.
3. The system of claim 1, wherein said system is part of a storage area network (SAN) controller.	Van Loo '485 teaches “[t]he system of claim 1, wherein said system is part of a storage area network (SAN) controller,” including at least the following exemplary citations. For example, Van Loo '485 teaches wherein said system is part of a storage area network (SAN) controller, by disclosing, as non-limiting examples, the first processing element (e.g., a master device, or a system controller) and the second processing element (e.g., a system controller, or a master device) controlling

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	downstream slave devices including storages like RAMs, ROMs, or disk drives. The first processing element, the second processing elements, and the input and output queues are used in a SAN controller. <i>See, e.g.:</i> “One or more conventional memories or other data storage devices 85 and one or more input/output (I/O) devices 95 forming part of the system 10 are provided for user interface, data output, etc.; these various slave devices may include RAM, ROM, disk drives, monitors, keyboards, track balls, printers, etc. They are coupled into the interconnect module 25 via the slave interfaces 80. The "slave" designation means in this case only that such devices accept requests from one or more processors and fulfill those requests.” Van Loo ’485 at col. 5, ll. 20–28. “The interconnection network may in general take the form of a number of different standard communication topologies that interconnect masters and slaves, such as a point-to-point link, a single bus or multiple buses, or switching fabrics. The interconnect may employ any of a number of conventional mechanisms for switching the transaction request to a slave using one or more signal paths, and the switching may be based either on the addressing information contained in the transaction request packet, or on another protocol not necessarily dependent on the contents of the request packet. There may be any amount of buffering, or no buffering, in the interconnect.” Van Loo ’485 at col. 5, ll. 30–41.
4. The system of claim 1, wherein said system is part of a network attached storage (NAS) controller.	Van Loo ’485 teaches “[t]he system of claim 1, wherein said system is part of a network attached storage (NAS) controller,” including at least the following exemplary citations. For example, Van Loo ’485 teaches wherein said system is part of a network attached storage (NAS) controller, by disclosing, as non-limiting examples, the first processing element (e.g., a master device, or a system controller) and the second processing element (e.g., a system controller, or a master device) controlling downstream slave devices including storages like RAMs, ROMs, or disk drives. The first processing element, the second processing elements, and the input and output queues are used in a NAS controller. <i>See, e.g.:</i> “One or more conventional memories or other data storage devices 85 and one or more input/output (I/O) devices 95 forming part of the system 10 are provided for user interface, data output, etc.; these various slave devices may include RAM, ROM, disk drives, monitors, keyboards, track balls, printers, etc. They are coupled into the interconnect module 25 via the slave interfaces 80. The "slave" designation means in this case only that such devices accept requests from one or more processors and fulfill those requests.” Van Loo ’485 at col. 5, ll. 20–28.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“The interconnection network may in general take the form of a number of different standard communication topologies that interconnect masters and slaves, such as a point-to-point link, a single bus or multiple buses, or switching fabrics. The interconnect may employ any of a number of conventional mechanisms for switching the transaction request to a slave using one or more signal paths, and the switching may be based either on the addressing information contained in the transaction request packet, or on another protocol not necessarily dependent on the contents of the request packet. There may be any amount of buffering, or no buffering, in the interconnect.” Van Loo ’485 at col. 5, ll. 30–41.
5. The system of claim 1, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	Van Loo ’485 teaches “[t]he system of claim 1, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element,” including at least the following exemplary citations. Van Loo ’485 supports at least two example theories for invalidity. In the first example theory, Van Loo ’485 teaches wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element by disclosing, as non-limiting examples, a counter on a system controller incrementing its value by a predefined value (e.g., 1) prior to returning said stored value to said at least one second processing element. A counter on a system controller decrements the stored value by a predefined value. In the second example theory, Van Loo ’485 teaches wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element by disclosing, as non-limiting examples, a master counter incrementing its value by a predefined value (e.g., 1) prior to returning said stored value to said at least one second processing element. A master counter decrements the stored value by a predefined value. <i>See, e.g.:</i>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420[New Transaction request 420] 420 --> 424[For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424] 424 -- No --> 426[Wait 426] 424 -- Yes --> 430[Is master counter value = total SCIQ size? 430] 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 Fig. 3B]) 430 -- Yes --> 500[Transaction complete message (S_REPLY) received from SC? 500] 500 -- No --> 510[Wait 510] 510 --> 424 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 440 420 --> 422[End 422] </pre> FIG. 3A Van Loo '485 at Figure 3A. “The masters and the controller are prevented from issuing any transaction requests (or to initiate a write-data transfer requests) downstream when the respective counter indicates that the corresponding request or data queue downstream is full. When a transaction is complete (e.g. upon the receipt of requested data read or consumption of write data by the slave), the relevant counter is decremented to indicate the availability of a place in the transaction queue or write-data queue.” Van Loo '485 at col. 3, ll. 7–15. “When a master, e.g. master interface 110, has a transaction request to issue, it first compares the value in its counter 310 with the value in the config register 270. If the counter value is less than the config register value, then the request may be issued. The request is sent from the master's output queue 290 to the SCIQ 250, and the counter 310 is incremented by one.” Van Loo '485 at col. 9, ll. 12–18. “The SC 130 then determines to which of the two slaves 140 and 150 the transaction request is destined, and checks the counter for that queue. For instance, if slave 140 is the destination for the transaction request, then the SC 130 checks the counter 210 and compares the value stored there with the value in the config register 200 corresponding to the ID register 180. If the counter 230 value is less than the value stored in the config register, then the SC 130 issues the transaction request and increments the counter 230. Otherwise, the

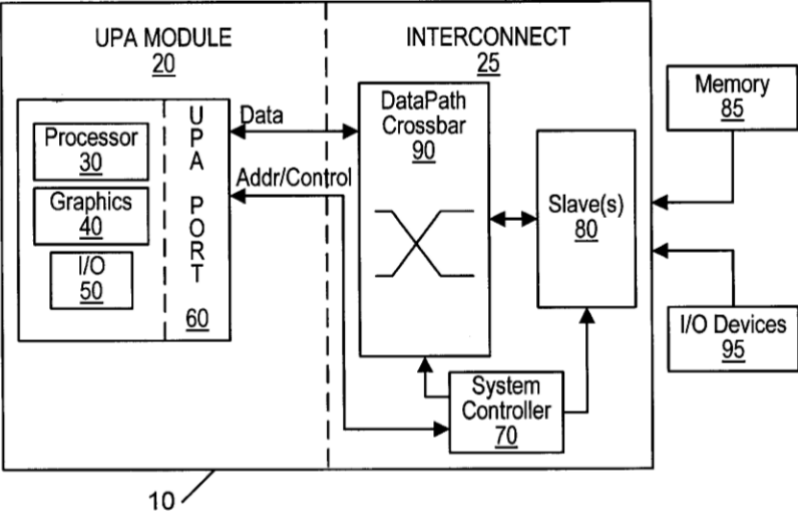
Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	transaction request is maintained in the output queue 210. (In some transactions related to ordering constraints for transactions for different requests from the same master, it may be desirable to leave the request in the SCIQ 250.)” Van Loo ’485 at col. 9, ll. 19–32. “Assuming the transaction request is issued in this example, then the SC 130 sends a signal to the master 110 to this effect (upon completion of the transaction, e.g. the transfer of data) and removes the transaction request from its input queue 250 (upon sending of the reply). The master 110 accordingly decrements its counter 310, which enables it to issue an additional pending transaction request. If the counter 310 was at its maximum (indicating that the SCIQ 250 was full), the decrementation of the counter 310 allows room for a single additional transaction request from the master 110 to the SC 130. If the counter 310 was not at its maximum value, then the decrementation of the counter 310 simply adds one to the number of transaction requests available to the master interface 110.” Van Loo ’485 at col. 9, ll. 33–46. “Since at start-up the config registers 200 fields and the config registers 270-280 must allow at least one transaction apiece (to read their corresponding ID registers 180-190 and 250-260, respectively), they are initialized to a value of "1" to begin with, to "bootstrap" the start-up. Then, when the read-ID-registers transaction requests are issued, and the counters are decremented, the requests will be allowed. (If the config registers were all at 0, no transactions would be allowed.) Upon the reading of the respective ID registers, the config register values are replaced with the correct values, i.e. the actual sizes of their associated ID registers.” Van Loo ’485 at col. 10, ll. 56–66. “When this complete-transaction signal is received by the master interface 110 (box 500), the master interface decrements the counter associated with that SCIQ (box 530) and proceeds to the step at box 440. At box 440, the counter 320 is incremented by one, and at box 450 the transaction request is sent by the master to the SC. Thus, the counter 320 now reflects the sending of one (or one additional) transaction request.” Van Loo ’485 at col. 11, ll. 39–46. “At box 475, the SC 130 then checks the counter for the recipient slave, e.g. counter 240 if slave 150 is the destination for the pending transaction request. If the counter equals or exceeds the value in the config register (i.e. the size indicated by the ID register 180 or 190, which were read at initialization), then the request is not yet allowed. In this case, then steps 530-550 are followed (essentially identical to steps 500-520), until a free line opens up in the destination slave queue to receive the transaction request.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	If the appropriate counter (230 or 240) has not reached its maximum allowed value, then it is incremented by one (box 480), and the transaction request is sent to the recipient slave (box 490).” Van Loo ’485 at col. 12, ll. 50–62. “In general in FIGS. 4-7, the order of events is indicated by the circled event numerals 1, 2, 3 and 4 (with accompanying arrows indicating the direction of data or signal flow), as the case may be for each figure. With the exception of the fact that the memories in FIGS. 4 and 5 do not include the slave queues and ID register of the slaves shown in FIG. 2, the following description of data flow with respect to FIGS. 4-7 should be understood to include the steps described with respect transaction request control (see FIGS. 3A-3B). Thus, for each request issued, the appropriate counter consultation, incrementation and decrementation is carried out to determine that the request is sent at an appropriate time. The respective queues are also handled as appropriate.” Van Loo ’485 at col. 13, l. 58 – col. 14, l. 3.
6. The system of claim 1, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Van Loo ’485 teaches “[t]he system of claim 1, wherein at least one third processing element executes said command and notifies said output queue to issue said response,” including at least the following exemplary citations. For example, Van Loo ’485 teaches wherein at least one third processing element executes said command and notifies said output queue to issue said response, by disclosing, as non-limiting examples, a slave device executing the transaction request and send a result (e.g., a transaction-completion signal or a S_REPLY) to the output queue. A slave device executes the command and notifies the output queue to issue a response. <i>See, e.g.:</i> “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo ’485 at Abstract.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1 Van Loo '485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture (FIG. 1A) centered around a DataPath Crossbar (92). The crossbar is connected to several components: Masters: Master M1, Master M2, and Master M3 (all labeled "can act as Slave") are connected to the crossbar via data lines D1, D2, and D3 respectively. They are also connected to the System Controller (75) via address/control lines A/ctrl-1, A/ctrl-2, and A/ctrl-3. boot PROM (94): Connected to the crossbar via data line D4. Memory and Slave Devices: Mem1, Mem2, S1, and S2 are connected to the crossbar via data lines D5, D6, D7 and address/control lines A/ctrl-4, A/ctrl-5, A/ctrl-6, and A/ctrl-7 respectively. System Controller (75): Connected to the crossbar via a DataPath Control line. FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The present invention uses a new system interconnect architecture and concomitant new methods for utilizing the interconnect to control transaction requests and data flow between master devices and slave or memory devices. In FIG. 1, the system 10 includes a UPA module 20 and an interconnect network or module 25, which in different embodiments of the invention may or may not be connected to the data path. The UPA module may include such devices as a processor 30, a graphics unit 40, and an I/O unit 50. Other units may be included, and act as the master units for the purposes of the present invention. A master interface is defined as the interface for any entity initiating transaction requests; examples of such masters are a CPU making memory requests, or an I/O channel and bridges making DMA requests.” Van Loo '485 at col. 4, ll. 27–41. “In general, in this application a master is exemplified by a processor. However, a master may be any transaction-requesting device, whether or not it includes a microprocessor. Similarly, a "slave" refers herein to any device that can accept a transaction request, including both memory and non-memory devices, etc., and including devices such as processors and I/O controllers that may themselves act as masters.” Van Loo '485 at col. 4, ll. 42–49.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“For the purposes of this invention, a "transaction" may be defined as a request packet issued by a master, followed by an acknowledgment packet (not necessarily a full packet, depending upon the chosen implementation) from the recipient immediately downstream. There may or may not be a data transfer accompanying a request packet, and the data transfer may either occur on the same set of wires as the request packet, or on separate datapath wires. This is described in greater detail below in connection with FIGS. 4-7.” Van Loo ’485 at col. 4, ll. 50–59. “One or more conventional memories or other data storage devices 85 and one or more input/output (I/O) devices 95 forming part of the system 10 are provided for user interface, data output, etc.; these various slave devices may include RAM, ROM, disk drives, monitors, keyboards, track balls, printers, etc. They are coupled into the interconnect module 25 via the slave interfaces 80. The "slave" designation means in this case only that such devices accept requests from one or more processors and fulfill those requests.” Van Loo ’485 at col. 5, ll. 20–28. “FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “Slave devices S1 . . . S2 (which may be one, several or many slave devices) are also provided, and the masters, memories and slaves are coupled via a system controller 75 in the same fashion as the system controller 70 is coupled to the master and slave(s) in FIG. 1. The SC 75 is coupled via a datapath control bus 77 to a datapath crossbar (or interconnect) 92, as with the datapath crossbar 90 in FIG. 1.” Van Loo ’485 at col. 5, ll. 61–67. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“In FIG. 2 the slaves 140 and 150 may be any of the slave devices described above with respect to slave 80 in FIG. 1, and in particular slaves 140 and 150 may represent any number of memory or non-memory slave devices.” Van Loo ’485 at col. 6, ll. 49–52. “If the pending transaction is a slave read request then at this point (box 495) the read operation is initiated. When it is complete, the slave sends a P.. REPLY to the SC, and the SC sends S.. REPLY's to both the requesting master and the recipient slave. See the discussion below relating to FIG. 6 below for details about the transaction and data flow for slave read requests.” Van Loo ’485 at col. 12, l. 63 – col. 13, l. 2. “Memory Read Requests: FIG. 4 This read request example assumes that data is coming from memory, and not, e.g., from a cache. Snoop operations on the snoopbus shown in FIG. 4 are not in consideration here. Event 1: When a UPA master port such as port 630 has a read-from-memory transaction ready, and the master counter is not at its allowed maximum (see box 430 in FIGS. 3A-3B), the read transaction is issued on the UPA.. Addressbus from UPA port 630 to the system controller 650. This is indicate by the event 1 (P.. REQ) along the UPA.. Addressbus 660 in FIG. 4, with the direction of the information indicated by the arrow, i.e. from the port to the SC. Event 2: The memory cycle i.e. RAS (read-address-strobe)/CAS (column-address-strobe) request issuance! is issued over memory control bus 670 to the memory banks 610 . . . 620. See vent 2 ("RAS/CAS") along bus 670. Event 3: The datapath is scheduled by a signal along the datapath control bus 680, and data items are accordingly delivered from memory to the datapath crossbar 625 via a memory databus 690 and UPA databus 700. This fulfills the read request.” Van Loo ’485 at col. 14, ll. 4–25. “Slave Read Requests: FIG. 6 FIG. 6 illustrates a read sequence to a slave device other than memory, and is similar to FIG. 2, but for this example a single master interface 710 and a single slave interface 720 are used, coupled by a system controller 720 and a datapath crossbar 730. Event 1 indicates the issuance of a read request P.. REQ on UPA address bus 740 to SC 720.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	In event 2, the SC 720 sends the P.. REQ on bus 750 to the slave interface 710. To do this, if there are several slave interfaces, the SC must first decode the address to ensure that the P.. REQ goes to the correct slave interface. Event 2 informs the slave interface to prepare the data to move through the datapath. When the data items are ready, then event 3 takes place, namely the sending of a P.. REPLY from the slave 710 to the SC 720 over bus 760. In event 4, a series of steps are executed to cause the master interface to receive the data: SC 720 schedules the datapath 730, and issues an S.. REPLY over bus 770 to the master interface 700. In addition, the SC issues the S.. REPLY over bus 780 to the slave 710, to drive the data, when it is ready, on the slave's UPA databus 790 via the datapath and over the databus 800 to the master interface 700.” Van Loo ’485 at col. 14, l. 49 – col. 15, l. 6.
[7pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	Van Loo ’485 teaches “[a] system for dividing and synchronizing a processing task across a plurality of processing elements,” including at least the following exemplary citations. <i>See above at row [1pre].</i>
[7a] an input queue implemented in hardware;	Van Loo ’485 teaches “an input queue implemented in hardware,” including at least the following exemplary citations. <i>See above at row [1a].</i>
[7b] an output queue implemented in hardware;	Van Loo ’485 teaches “an output queue implemented in hardware,” including at least the following exemplary citations. <i>See above at row [1b].</i>
[7c] a first processing element having access to said input queue and said output queue;	Van Loo ’485 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations. <i>See above at row [1c].</i>

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[7d] at least one second processing element in communication with said input queue;	Van Loo '485 teaches “at least one second processing element in communication with said input queue,” including at least the following exemplary citations. For example, Van Loo '485 teaches at least one second processing element in communication with said input queue, by disclosing, as non-limiting examples, a system controller interacting with the input queue. A system controller, the second processing element is in communication with the input queue. <i>See, e.g.:</i> “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo '485 at Abstract.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 1 Van Loo '485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture centered around a DataPath Crossbar (92). On the left, three Master blocks (M1, M2, M3) are shown, each with the note "(can act as Slave)". They are connected to the crossbar via data paths D1, D2, and D3. A boot PROM (94) is also connected to the crossbar via data path D4. On the right, two Memory blocks (Mem1, Mem2) and two Slave blocks (S1, S2) are connected to the crossbar via data paths D4, D5, D6, and D7. A System Controller (75) is connected to the crossbar via a DataPath Control line and several control lines (A/ctrl-1 to A/ctrl-7). The crossbar is represented by a box with a cross symbol and the number 92. The boot PROM is labeled 94. The System Controller is labeled 75. The data paths are labeled D1 through D7. The control lines are labeled A/ctrl-1 through A/ctrl-7. The crossbar is also labeled 96. FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Includes Logic (110), MASTER QUEUE (112), Config Register (270), and Counter (310). Master 2 (125): Includes Logic (122), MASTER QUEUE (123), Config Register (280), and Counter (320). Slave 1 (140): Includes Logic (142), Slave Queue (185), ID Register (188), and Counter (190). Slave 2 (150): Includes Logic (152), Slave Queue (195), ID Register (198), and Counter (170). System Controller (130): Includes SC LOGIC (245), SCID Register (260), SCID Register (250), SCID Register (255), SCID Register (258), SCID Register (265), SCID Register (270), SCID Register (275), SCID Register (280), SCID Register (285), SCID Register (290), SCID Register (295), SCID Register (300), SCID Register (305), SCID Register (310), SCID Register (315), SCID Register (320), SCID Register (325), SCID Register (330), SCID Register (335), SCID Register (340), SCID Register (345), SCID Register (350), SCID Register (355), SCID Register (360), SCID Register (365), SCID Register (370), SCID Register (375), SCID Register (380), SCID Register (385), SCID Register (390), SCID Register (395), SCID Register (400), SCID Register (405), SCID Register (410), SCID Register (415), SCID Register (420), SCID Register (425), SCID Register (430), SCID Register (435), SCID Register (440), SCID Register (445), SCID Register (450), SCID Register (455), SCID Register (460), SCID Register (465), SCID Register (470), SCID Register (475), SCID Register (480), SCID Register (485), SCID Register (490), SCID Register (495), SCID Register (500), SCID Register (505), SCID Register (510), SCID Register (515), SCID Register (520), SCID Register (525), SCID Register (530), SCID Register (535), SCID Register (540), SCID Register (545), SCID Register (550), SCID Register (555), SCID Register (560), SCID Register (565), SCID Register (570), SCID Register (575), SCID Register (580), SCID Register (585), SCID Register (590), SCID Register (595), SCID Register (600), SCID Register (605), SCID Register (610), SCID Register (615), SCID Register (620), SCID Register (625), SCID Register (630), SCID Register (635), SCID Register (640), SCID Register (645), SCID Register (650), SCID Register (655), SCID Register (660), SCID Register (665), SCID Register (670), SCID Register (675), SCID Register (680), SCID Register (685), SCID Register (690), SCID Register (695), SCID Register (700), SCID Register (705), SCID Register (710), SCID Register (715), SCID Register (720), SCID Register (725), SCID Register (730), SCID Register (735), SCID Register (740), SCID Register (745), SCID Register (750), SCID Register (755), SCID Register (760), SCID Register (765), SCID Register (770), SCID Register (775), SCID Register (780), SCID Register (785), SCID Register (790), SCID Register (795), SCID Register (800), SCID Register (805), SCID Register (810), SCID Register (815), SCID Register (820), SCID Register (825), SCID Register (830), SCID Register (835), SCID Register (840), SCID Register (845), SCID Register (850), SCID Register (855), SCID Register (860), SCID Register (865), SCID Register (870), SCID Register (875), SCID Register (880), SCID Register (885), SCID Register (890), SCID Register (895), SCID Register (900), SCID Register (905), SCID Register (910), SCID Register (915), SCID Register (920), SCID Register (925), SCID Register (930), SCID Register (935), SCID Register (940), SCID Register (945), SCID Register (950), SCID Register (955), SCID Register (960), SCID Register (965), SCID Register (970), SCID Register (975), SCID Register (980), SCID Register (985), SCID Register (990), SCID Register (995). DataPath (155): Connects Master 1 and Master 2 to Slave 1 and Slave 2. DataPath Control (156): Connects Master 1 and Master 2 to Slave 1 and Slave 2. I/O Devices & Memory (95): Connects to the System Controller. 100 FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a memory write flow control system. It features multiple UPA ports (630, 640, ..., 640) connected to a central Datapath Crossbar (625). The crossbar is also connected to a Memory Databus (690) and a Memory Bank (610). A System Controller & Address Crossbar (650) is connected to the crossbar and a Coherence/Memory/Datapath Control block (660). The control block is also connected to a Memory Control block (670). A Snoobus (600) is connected to the system. The diagram includes various control signals and data paths, such as P_REQ, S_REPLY, P_REPLY, Addr_Valid, Data_Stall, ECC_Valid, Arb_Cnt, Port_ID, Sys_Clk, and data. It also shows a Memory Write Flow Control block (600) and a Diag Control block. FIG. 5 Van Loo '485 at Figure 5.

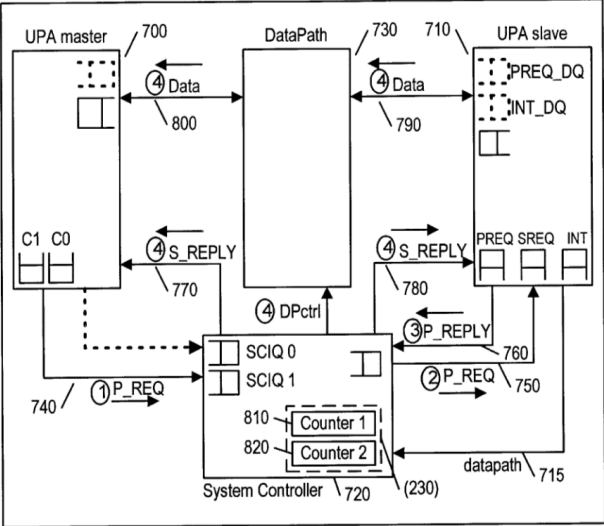
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 - col. 3, l. 6. “The present invention uses a new system interconnect architecture and concomitant new methods for utilizing the interconnect to control transaction requests and data flow between master devices and slave or memory devices.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	In FIG. 1, the system 10 includes a UPA module 20 and an interconnect network or module 25, which in different embodiments of the invention may or may not be connected to the data path. The UPA module may include such devices as a processor 30, a graphics unit 40, and an I/O unit 50. Other units may be included, and act as the master units for the purposes of the present invention. A master interface is defined as the interface for any entity initiating transaction requests; examples of such masters are a CPU making memory requests, or an I/O channel and bridges making DMA requests.” Van Loo ’485 at col. 4, ll. 27–41. “In general, in this application a master is exemplified by a processor. However, a master may be any transaction-requesting device, whether or not it includes a microprocessor. Similarly, a "slave" refers herein to any device that can accept a transaction request, including both memory and non-memory devices, etc., and including devices such as processors and I/O controllers that may themselves act as masters.” Van Loo ’485 at col. 4, ll. 42–49. “In a preferred embodiment, the system controller 70 and UPA interface 60 are carried on the main processor chip, and the slave interface is on the motherboard, though many variations are possible. More generally, each master (whether a processor or some other device) has a UPA master interface, and each slave includes a UPA slave interface. The system controller in each case resides with the system.” Van Loo ’485 at col. 5, ll. 4-11. “A datapath crossbar 90 is also included in the interconnect module 25, and is coupled to the slave interface(s), the system controller 70, and the ports 60. The datapath crossbar may be a simple bus or may be a more complicated crossbar. (The UPA ports 60 may be configured as part of either the UPA module 20 or the interconnect module 25.) The datapath unit 90 is used to transmit read and write data in a manner to be described below.” Van Loo ’485 at col. 5, ll. 12–19. “FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31. “Each slave 140 and 150 includes a slave queue (160 and 170, respectively) for receiving transaction requests. The maximum sizes of these slave queues are represented by values in port ID registers 180 and 190, respectively.” Van Loo ’485 at col. 6, ll. 32–35. “Masters 110 and 120 include data queues or buffers 115 and 125, and slaves 140 and 150 include data queues or buffers 185 and 195, whose functions are described in detail relative to FIGS. 6 and 7 below. The maximum sizes of the slave write data queues 185 and 195 are also represented by values in port ID registers 180 and 190, respectively. In the special case where there is a one-to-one correspondence to a request queue entry (e.g. 100) and a data buffer in the write data queue (e.g. 185), with the write data queue being maximally dimensioned to hold an entire packet (i.e. dimensioned such that it can hold the largest contemplated packet size), then the queue size in 180 can be represented by a single number.” Van Loo ’485 at col. 6, ll. 36–48. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[7e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;	Van Loo ’485 teaches “a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element,” including at least the following exemplary citations. <i>See above at row [1e].</i>
[7f] computer storage storing instructions, which when executed by said at least one second processing element:	Van Loo ’485 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations. <i>See above at row [1f].</i>
[7g] receives a command from said first processing element;	Van Loo ’485 teaches “receives a command from said first processing element,” including at least the following exemplary citations. For example, Van Loo ’485 teaches that the second processing element executes instructions that causes it to receive a command from said first processing element, by disclosing, as non-limiting examples, the second processing element (e.g., a system controller) receiving a transaction request from a master device. The second processing element receives a command from the first processing element. <i>See, e.g.:</i>

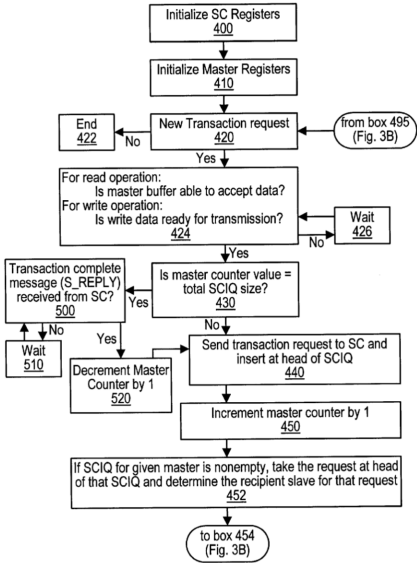
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420[New Transaction request 420] 495([from box 495 (Fig. 3B)]) --> 420 420 --> 424[For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424] 424 -- No --> 426[Wait 426] 426 --> 424 424 -- Yes --> 430[Is master counter value = total SCIQ size? 430] 430 -- Yes --> 500[Transaction complete message (S_REPLY) received from SC? 500] 500 -- No --> 510[Wait 510] 510 --> 500 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 440[Send transaction request to SC and insert at head of SCIQ 440] 430 -- No --> 440 440 --> 450[Increment master counter by 1 450] 450 --> 452[If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452] 452 --> 454([to box 454 (Fig. 3B)]) </pre> FIG. 3A Van Loo '485 at Figure 3A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 454} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision1 -- "(Slave operation)" --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] ReadWrite --> DetermineSlave[Determine recipient slave for transaction request 462] DetermineSlave --> Decision3 Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> MoveSCOQ[Move transaction onto SCOQ for the recipient slave 468] MoveSCOQ --> Note470[If this is a slave write transaction: SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 476} Decision4 -- Yes --> CompleteMsg[Transaction complete message received from slave? 480] CompleteMsg -- No --> Wait480[Wait 480] Wait480 --> CompleteMsg CompleteMsg -- Yes --> DecSC[Decrement SC counter for recipient slave 482] DecSC --> Decision4 Decision4 -- No --> IncSC[Increment SC counter for recipient slave by 1 484] IncSC --> SendReq[Send transaction request to recipient slave 486] SendReq --> Note496[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 496] Note496 --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “When a master, e.g. master interface 110, has a transaction request to issue, it first compares the value in its counter 310 with the value in the config register 270. If the counter value is less than the config register value, then the request may be issued. The request is sent from the master's output queue 290 to the SCIQ 250, and the counter 310 is incremented by one.” Van Loo '485 at col. 9, ll. 12–18. “At box 420, it is determined whether a new transaction request is pending in one of the masters, e.g. the master 120.” Van Loo '485 at col. 10, ll. 67–68. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[7h] accesses said reservation register and reads said stored value;	Van Loo ’485 teaches “accesses said reservation register and reads said stored value,” including at least the following exemplary citations. <i>See</i> above at row [1g].
[7i] determines when said read value indicates available space in said input queue to issue said command;	Van Loo ’485 teaches “determines when said read value indicates available space in said input queue to issue said command,” including at least the following exemplary citations. <i>See</i> above at row [1h].
[7j] issues said command to said input queue on behalf of said first processing element; and	Van Loo ’485 teaches “issues said command to said input queue on behalf of said first processing element,” including at least the following exemplary citations. For example, Van Loo ’485 teaches that the second processing element executes instructions that causes it to issue said command to said input queue on behalf of said first processing element, by disclosing, as non-limiting examples, the second processing element (e.g., a system controller) issues the received transaction request into the input queue. The second processing element issues the command to the input queue on behalf of the first processing element. <i>See, e.g.:</i>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <pre> graph TD 400[Initialize SC Registers 400] --> 410[Initialize Master Registers 410] 410 --> 420{New Transaction request 420} 420 -- from box 495 (Fig. 3B) --> 420 420 -- No --> 422[End 422] 420 -- Yes --> 424{For read operation: Is master buffer able to accept data? For write operation: Is write data ready for transmission? 424} 424 -- No --> 426[Wait 426] 426 --> 424 424 -- Yes --> 430{Is master counter value = total SCIQ size? 430} 430 -- Yes --> 500{Transaction complete message (S_REPLY) received from SC? 500} 500 -- Yes --> 520[Decrement Master Counter by 1 520] 520 --> 450[Increment master counter by 1 450] 500 -- No --> 450 430 -- No --> 440[Send transaction request to SC and insert at head of SCIQ 440] 440 --> 450 450 --> 452{If SCIQ for given master is nonempty, take the request at head of that SCIQ and determine the recipient slave for that request 452} 452 --> 454([to box 454 (Fig. 3B)]) </pre> FIG. 3A Van Loo '485 at Figure 3A.

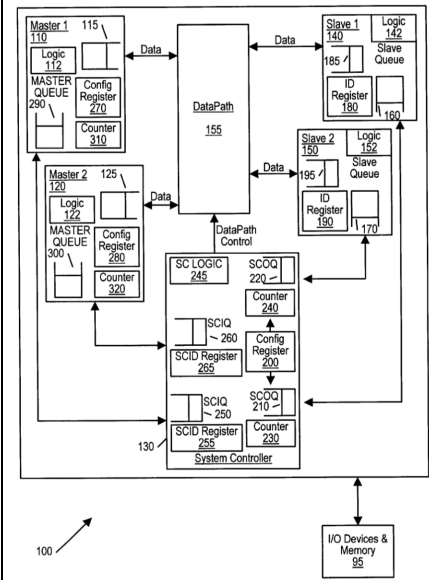
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<pre>graph TD Start([from box 452 (Fig. 3A)]) --> Decision1{Is this a memory operation or a slave operation? 554} Decision1 -- "(Memory operation)" --> Decision2{Is the recipient memory valid, given ordering constraints? 456} Decision2 -- No --> Wait458[Wait until memory is available 458] Wait458 --> Decision2 Decision2 -- Yes --> ReadWrite[Read or write memory as required and send S_REPLY 460] Decision1 -- "(Slave operation)" --> Determine462[Determine recipient slave for transaction request 462] Determine462 --> Decision3{is the recipient slave valid, given ordering constraints? 464} Decision3 -- No --> Wait466[Wait until slave is valid 466] Wait466 --> Decision3 Decision3 -- Yes --> Move468[Move transaction onto SCOQ for the recipient slave 468] Move468 --> Note470[If this is a slave write transaction, SC enables datapath, whereupon master moves data through data crossbar to slave input queue, and SC sends S_REPLY (to master and slave) 470] Note470 --> Decision4{Is SC counter value for recipient slave = P_REQ queue of recipient slave? 475} Decision4 -- No --> Increment480[Increment SC counter for recipient slave by 1 480] Increment480 --> Send482[Send transaction request to recipient slave 482] Send482 --> Note495[If this is a slave read request, then initiate read operation; when complete, slave sends P_REPLY to SC and SC sends S_REPLY's (to master and slave) 495] Note495 --> Decision4 Decision4 -- Yes --> Decrement500[Decrement SC counter for recipient slave 500] Decrement500 --> Wait540[Wait 540] Wait540 --> Complete530{Transaction complete message received from slave? 530} Complete530 -- No --> Wait540 Complete530 -- Yes --> End([to box 420 (Fig. 3A)])</pre> FIG. 3B Van Loo '485 at Figure 3B.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	“Assuming the transaction request is issued in this example, then the SC 130 sends a signal to the master 110 to this effect (upon completion of the transaction, e.g. the transfer of data) and removes the transaction request from its input queue 250 (upon sending of the reply). The master 110 accordingly decrements its counter 310, which enables it to issue an additional pending transaction request. If the counter 310 was at its maximum (indicating that the SCIQ 250 was full), the decrementation of the counter 310 allows room for a single additional transaction request from the master 110 to the SC 130. If the counter 310 was not at its maximum value, then the decrementation of the counter 310 simply adds one to the number of transaction requests available to the master interface 110.” Van Loo ’485 at col. 9, ll. 33–46. “At box 424, if the pending transaction request is for a read operation, then the system determines whether the master read data buffer (discussed in greater detail below) for the master interface is ready to accept data, i.e. whether there is sufficient room in the master read data buffer to receive the data to be read. If not, then the system waits as at box 426 until the master read data buffer is ready. Note that a write operation need not be held up during a wait period for a read operation, but may proceed independently; and vice versa. For a write operation, the system determines whether the data to be written to one of the slaves via a slave interface or memory is in fact ready for writing in (transmission to) a master write buffer. If not, again at box 426 a wait period is executed, until the data item is ready for writing.” Van Loo ’485 at col. 11, ll. 4–17. “The SC is provided with intelligence and/or logic (hardware and/or software) to determine whether it has a transaction request pending in its request receive queue (such as SCIQ’s 250 and 260). If so, then at box 452 the transaction request at the head of the queue is examined to determine which slave is intended as the recipient for the request. This queue control and recipient determination is carried out in a conventional manner.” Van Loo ’485 at col. 11, ll. 55–62. “Once the slave is valid for this transaction, then the transaction request is moved into the corresponding SC output queue (SCOQ) 210 or 220.” Van Loo ’485 at col. 12, ll. 39–41. “If the pending transaction is a slave write transaction, than at this time (box 470) the SC enables the datapath 155 via a datapath control signal, and the master (whose transaction is pending) is then able to move the data through the datapath to the appropriate slave input queue (185 or 195). The SC then sends its transaction-complete (S- REPLY) signal to both the master and the slave (see discussion below relative to FIG. 7).” Van Loo ’485 at col. 12, ll. 42–49. “At box 475, the SC 130 then checks the counter for the recipient slave, e.g. counter 240 if slave 150 is the destination for the pending transaction request. If the counter equals or exceeds the value in the config

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	register (i.e. the size indicated by the ID register 180 or 190, which were read at initialization), then the request is not yet allowed. In this case, then steps 530-550 are followed (essentially identical to steps 500-520), until a free line opens up in the destination slave queue to receive the transaction request. If the appropriate counter (230 or 240) has not reached its maximum allowed value, then it is incremented by one (box 480), and the transaction request is sent to the recipient slave (box 490).” Van Loo ’485 at col. 12, ll. 50–62. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device; (7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[7k] wherein said first processing element receives a response corresponding to said command from said output queue.	Van Loo ’485 teaches “wherein said first processing element receives a response corresponding to said command from said output queue,” including at least the following exemplary citations. For example, Van Loo ’485 teaches wherein said first processing element receives a response corresponding to said command from said output queue, by disclosing, as non-limiting examples, the first processing element (e.g., a master device) receiving a transaction-completion signal or a S_REPLY corresponding to a

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	transaction request. The first processing element receives a response from the output queue corresponding to the command. <i>See, e.g.:</i>  FIG. 2 Van Loo '485 at Figure 2.

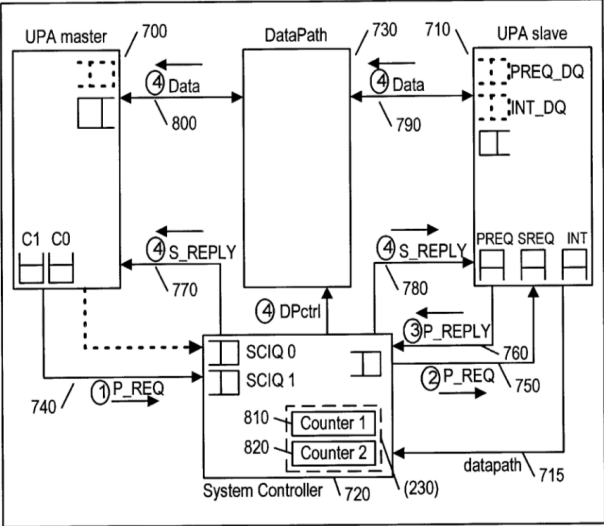
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “For the purposes of this invention, a "transaction" may be defined as a request packet issued by a master, followed by an acknowledgment packet (not necessarily a full packet, depending upon the chosen implementation) from the recipient immediately downstream. There may or may not be a data transfer accompanying a request packet, and the data transfer may either occur on the same set of wires as the request packet, or on separate datapath wires. This is described in greater detail below in connection with FIGS. 4-7.” Van Loo '485 at col. 4, ll. 50–59. “FIG. 6: read operation from a device other than memory;” Van Loo '485 at col. 13, l. 21. “Memory Read Requests: FIG. 4 This read request example assumes that data is coming from memory, and not, e.g., from a cache. Snoop operations on the snoopbus shown in FIG. 4 are not in consideration here. Event 1: When a UPA master port such as post 630 has a read-from-memory transaction ready, and the master counter is not at its allowed maximum (see box 430 in FIGS. 3A-3B), the read transaction is issued on the UPA.. Addressbus from UPA port 630 to the system controller 650. This is indicate by the event 1 (P..

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	REQ) along the UPA.. Addressbus 660 in FIG. 4, with the direction of the information indicated by the arrow, i.e. from the port to the SC. Event 2: The memory cycle i.e. RAS (read-address-strobe)/CAS (column-address-strobe) request issuance! is issued over memory control bus 670 to the memory banks 610 . . . 620. See vent 2 ("RAS/CAS") along bus 670. Event 3: The datapath is scheduled by a signal along the datapath control bus 680, and data items are accordingly delivered from memory to the datapath crossbar 625 via a memory databus 690 and UPA databus 700. This fulfills the read request.” Van Loo ’485 at col. 14, ll. 4–25. “Slave Read Requests: FIG. 6 FIG. 6 illustrates a read sequence to a slave device other than memory, and is similar to FIG. 2, but for this example a single master interface 710 and a single slave interface 720 are used, coupled by a system controller 720 and a datapath crossbar 730. Event 1 indicates the issuance of a read request P.. REQ on UPA address bus 740 to SC 720. In event 2, the SC 720 sends the P.. REQ on bus 750 to the slave interface 710. To do this, if there are several slave interfaces, the SC must first decode the address to ensure that the P.. REQ goes to the correct slave interface. Event 2 informs the slave interface to prepare the data to move through the datapath. When the data items are ready, then event 3 takes place, namely the sending of a P.. REPLY from the slave 710 to the SC 720 over bus 760. In event 4, a series of steps are executed to cause the master interface to receive the data: SC 720 schedules the datapath 730, and issues an S.. REPLY over bus 770 to the master interface 700. In addition, the SC issues the S.. REPLY over bus 780 to the slave 710, to drive the data, when it is ready, on the slave's UPA databus 790 via the datapath and over the databus 800 to the master interface 700.” Van Loo ’485 at col. 14, l. 49 – col. 15, l. 6.
8. The system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer.	Van Loo ’485 teaches “[t]he system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer,” including at least the following exemplary citations. <i>See above at row 2.</i>

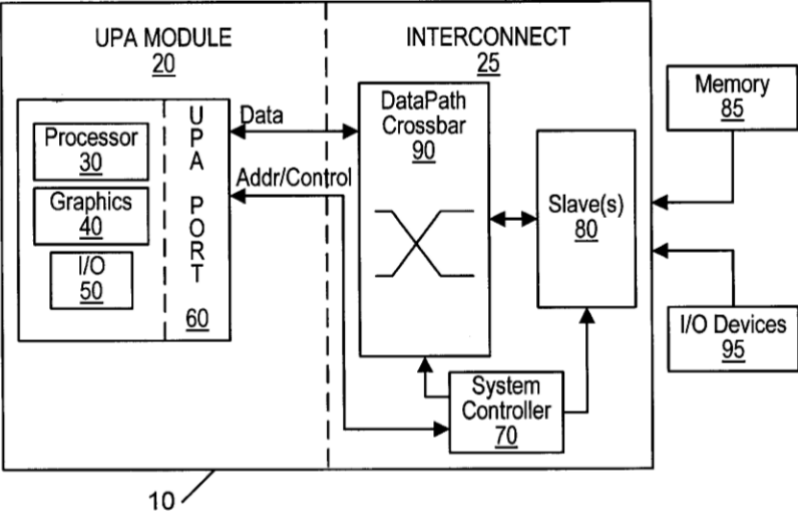
Exhibit 596-17

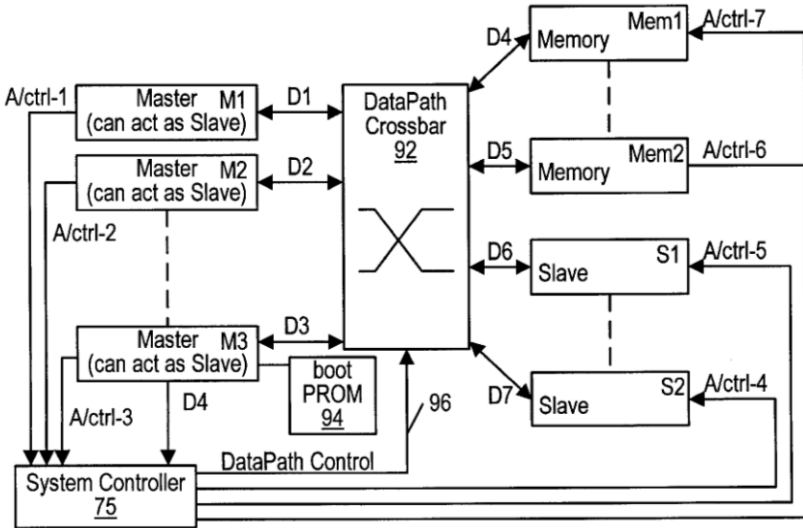
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
9. The system of claim 7, wherein said system is part of a storage area network (SAN) controller.	Van Loo '485 teaches “[t]he system of claim 7, wherein said system is part of a storage area network (SAN) controller,” including at least the following exemplary citations. <i>See above at row 3.</i>
10. The system of claim 7, wherein said system is part of a network attached storage (NAS) controller.	Van Loo '485 teaches “[t]he system of claim 7, wherein said system is part of a network attached storage (NAS) controller,” including at least the following exemplary citations. <i>See above at row 4.</i>
11. The system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	Van Loo '485 teaches “[t]he system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element,” including at least the following exemplary citations. <i>See above at row 5.</i>
12. The system of claim 7, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Van Loo '485 teaches “[t]he system of claim 7, wherein at least one third processing element executes said command and notifies said output queue to issue said response,” including at least the following exemplary citations. <i>See above at row 6.</i>
[13pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	Van Loo '485 teaches “[a] system for dividing and synchronizing a processing task across a plurality of processing elements,” including at least the following exemplary citations. <i>See above at row [1pre].</i>
[13a] an input queue implemented in hardware;	Van Loo '485 teaches “an input queue implemented in hardware,” including at least the following exemplary citations. <i>See above at row [1a].</i>

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[13b] an output queue implemented in hardware;	Van Loo '485 teaches “an output queue implemented in hardware,” including at least the following exemplary citations. <i>See</i> above at row [1b].
[13c] a first processing element having access to said input queue and said output queue;	Van Loo '485 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations. <i>See</i> above at row [1c].
[13d] at least one second processing element in communication with said input queue and said output queue;	Van Loo '485 teaches “at least one second processing element in communication with said input queue and said output queue,” including at least the following exemplary citations. For example, Van Loo '485 teaches at least one second processing element in communication with said input queue and said output queue, by disclosing, as non-limiting examples, the second processing element (e.g., a system controller) interacting with the input and output queues. The second processing element is in communication with the input and output queue. <i>See, e.g.:</i> “This invention describes a link-by-link flow control method for packet-switched uniprocessor and multiprocessor computer systems that maximizes system resource utilization and throughput, and minimizes system latency. The computer system comprises one or more master interfaces, one or more slave interfaces, and an interconnect system controller which provides dedicated transaction request queues for each master interface and controls the forwarding of transactions to each slave interface. The master interface keeps track of the number of requests in the dedicated queue in the system controller, and the system controller keeps track of the number of requests in each slave interface queue. Both the master interface, and system controller know the maximum capacity of the queue immediately downstream from it, and does not issue more transaction requests than what the downstream queue can accommodate. An acknowledgment from the downstream queue indicates to the sender that there is space in it for another transaction. Thus no system resources are wasted trying to send a request to a queue that is already full.” Van Loo '485 at Abstract.

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1 Van Loo '485 at Figure 1.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 FIG. 1A Van Loo '485 at Figure 1A.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	The diagram illustrates a system architecture with the following components and connections: Master 1 (115): Contains Logic 110, MASTER QUEUE 112, Config Register 270, and Counter 310. Master 2 (125): Contains Logic 122, MASTER QUEUE 123, Config Register 280, and Counter 320. Slave 1 (140): Contains Logic 142, Slave Queue 185, ID Register 188, and Counter 189. Slave 2 (150): Contains Logic 152, Slave Queue 195, ID Register 190, and Counter 191. System Controller (130): Contains SC LOGIC 245, SCIO ~ 260, SCID Register 265, SCIO ~ 250, SCIO ~ 210, SCID Register 255, Counter 240, Config Register 200, Counter 230, and SCIO ~ 250. DataPath (155): A central data path connecting Master 1, Master 2, Slave 1, and Slave 2. DataPath Control (195): A control unit for the data path. I/O Devices & Memory (95): Connected to the System Controller. Arrows indicate data flow and control signals between these components. 100 FIG. 2 Van Loo '485 at Figure 2.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	FIG. 4 Van Loo '485 at Figure 4.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “The present invention employs a method and apparatus for determining the total queue sizes of all the queues in the system at initialization time, and for permitting a master (e.g. a processor) to send a number of transaction requests only to the extent of that total. The system of the invention classifies system interconnect queues as request queues, read-data queues, and write-data queues, and determines rules of transfer of both requests and data as loosely coupled events. An interconnect (system) controller is connected between one or more masters (e.g. microprocessors) and the slave devices, which may be I/O units, disk drives, memory, etc. The interconnect controller includes a queue for each master, and each master includes a transaction counter indicating the number of outstanding transaction requests from that master to the controller. The interconnect controller additionally includes both request and write data queues for each downstream slave, and a transaction counter indicating the number of outstanding requests from the controller to that slave and the outstanding write data transfers from some master to that slave.” Van Loo '485 at col. 2, l. 54 – col. 3, l. 6. “The present invention uses a new system interconnect architecture and concomitant new methods for utilizing the interconnect to control transaction requests and data flow between master devices and slave or memory devices.

Exhibit 596-17

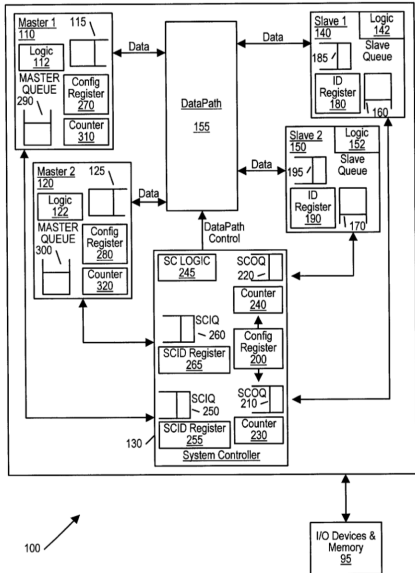
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	In FIG. 1, the system 10 includes a UPA module 20 and an interconnect network or module 25, which in different embodiments of the invention may or may not be connected to the data path. The UPA module may include such devices as a processor 30, a graphics unit 40, and an I/O unit 50. Other units may be included, and act as the master units for the purposes of the present invention. A master interface is defined as the interface for any entity initiating transaction requests; examples of such masters are a CPU making memory requests, or an I/O channel and bridges making DMA requests.” Van Loo ’485 at col. 4, ll. 27–41. “In general, in this application a master is exemplified by a processor. However, a master may be any transaction-requesting device, whether or not it includes a microprocessor. Similarly, a "slave" refers herein to any device that can accept a transaction request, including both memory and non-memory devices, etc., and including devices such as processors and I/O controllers that may themselves act as masters.” Van Loo ’485 at col. 4, ll. 42–49. “In a preferred embodiment, the system controller 70 and UPA interface 60 are carried on the main processor chip, and the slave interface is on the motherboard, though many variations are possible. More generally, each master (whether a processor or some other device) has a UPA master interface, and each slave includes a UPA slave interface. The system controller in each case resides with the system.” Van Loo ’485 at col. 5, ll. 4–11. “A datapath crossbar 90 is also included in the interconnect module 25, and is coupled to the slave interface(s), the system controller 70, and the ports 60. The datapath crossbar may be a simple bus or may be a more complicated crossbar. (The UPA ports 60 may be configured as part of either the UPA module 20 or the interconnect module 25.) The datapath unit 90 is used to transmit read and write data in a manner to be described below.” Van Loo ’485 at col. 5, ll. 12–19. “FIG. 1A shows a more generalized block diagram of a system according to the invention. Here, there are multiple masters (three exemplary masters M1-M3 being shown). These masters may act in certain circumstances as slaves. For instance, if M1 is a processor and M3 is an I/O controller, then M3 will often act as a slave to M1, as in the initialization procedure described below. On the other hand, during a DMA (direct memory access) operation, the I/O controller M3 will act as a master to a memory device, such as any of one to many of memories represented as M1 . . . M2 in FIG. 1A.” Van Loo ’485 at col. 5, ll. 50–60. “FIG. 2 illustrates an interconnect module 100 in a specific implementation where two master interfaces (or "masters") 110 and 120, a single system controller (SC) 130, two slaves interfaces (or "slaves") 140 and 150, and a datapath crossbar 155 are used. There may in principle any number of masters and slaves. The masters

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	may be any of the interfaces discussed above, or in general any devices or entities capable of issuing transaction requests.” Van Loo ’485 at col. 6, ll. 24–31. “Each slave 140 and 150 includes a slave queue (160 and 170, respectively) for receiving transaction requests. The maximum sizes of these slave queues are represented by values in port ID registers 180 and 190, respectively.” Van Loo ’485 at col. 6, ll. 32–35. “Masters 110 and 120 include data queues or buffers 115 and 125, and slaves 140 and 150 include data queues or buffers 185 and 195, whose functions are described in detail relative to FIGS. 6 and 7 below. The maximum sizes of the slave write data queues 185 and 195 are also represented by values in port ID registers 180 and 190, respectively. In the special case where there is a one-to-one correspondence to a request queue entry (e.g. 100) and a data buffer in the write data queue (e.g. 185), with the write data queue being maximally dimensioned to hold an entire packet (i.e. dimensioned such that it can hold the largest contemplated packet size), then the queue size in 180 can be represented by a single number.” Van Loo ’485 at col. 6, ll. 36–48. “6. A method for controlling transaction flow in a computer system including at least one master device, at least one slave device, and a system interconnect connected between the master and slave devices, the interconnect comprising a circuit separate from said master device and having at least one interconnect request queue coupled to each said master device and each said slave device including at least one slave request queue coupled to said system interconnect, the method including the steps of: (1) transmitting a first transaction request from a first said master device to a first said system interconnect; (2) receiving the first transaction request in a first said interconnect receive queue of said first system interconnect; (3) determining a recipient slave device for said first transaction request; (4) determining whether the recipient slave request queue of the recipient slave device is available to accept said first transaction request, and if so, forwarding said first transaction request to said recipient slave request queue; (5) receiving said first transaction request at said recipient slave request queue; (6) executing said first transaction request by said recipient slave device;

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	(7) notifying the system interconnect, by the slave device, that said first transaction request has been executed.” Van Loo ’485 at col. 17, ll. 33–60.
[13e] a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element;	Van Loo ’485 teaches “a reservation register implemented in hardware storing a value indicative of available space in said input queue, said reservation register accessible by both said first processing element and said at least one second processing element,” including at least the following exemplary citations. <i>See above at row [1e].</i>
[13f] computer storage storing instructions, which when executed by said at least one second processing element:	Van Loo ’485 teaches computer storage storing instructions that are executed by said at least one second processing element including at least the following exemplary citations. <i>See above at row [1f].</i>
[13g] receives a command from said first processing element;	Van Loo ’485 teaches “receives a command from said first processing element,” including at least the following exemplary citations. <i>See above at row [7g].</i>
[13h] accesses said reservation register and reads said stored value;	Van Loo ’485 teaches “accesses said reservation register and reads said stored value,” including at least the following exemplary citations. <i>See above at row [1g].</i>
[13i] determines when said read value indicates available space in said input queue to issue said command;	Van Loo ’485 teaches “determines when said read value indicates available space in said input queue to issue said command,” including at least the following exemplary citations. <i>See above at row [1h].</i>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[13j] issues said command to said input queue on behalf of said first processing element; and	Van Loo '485 teaches “issues said command to said input queue on behalf of said first processing element,” including at least the following exemplary citations. See above at row [7j].
[13k] receives a response corresponding to said command from said output queue.	Van Loo '485 teaches “receives a response corresponding to said command from said output queue,” including at least the following exemplary citations. For example, Van Loo '485 teaches that the second processing element executes instructions that causes it to receive a response corresponding to said command from said output queue, by disclosing, as non-limiting examples, the first processing element (e.g., a master device) receiving a transaction-completion response or a P_REPLY. The second processing element receives a corresponding response from the output queue.  FIG. 2 Van Loo '485 at Figure 2.

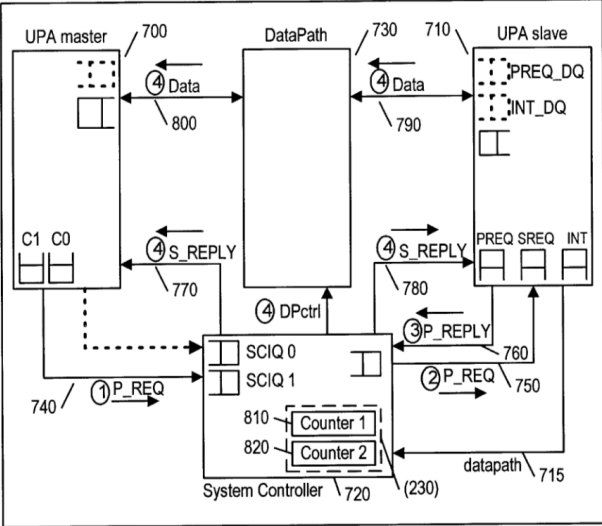
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 Slave Read Flow Control FIG. 6 Van Loo '485 at Figure 6. “For the purposes of this invention, a "transaction" may be defined as a request packet issued by a master, followed by an acknowledgment packet (not necessarily a full packet, depending upon the chosen implementation) from the recipient immediately downstream. There may or may not be a data transfer accompanying a request packet, and the data transfer may either occur on the same set of wires as the request packet, or on separate datapath wires. This is described in greater detail below in connection with FIGS. 4-7.” Van Loo '485 at col. 4, ll. 50–59. “FIG. 6: read operation from a device other than memory;” Van Loo '485 at col. 13, l. 21. “Memory Read Requests: FIG. 4 This read request example assumes that data is coming from memory, and not, e.g., from a cache. Snoop operations on the snoopbus shown in FIG. 4 are not in consideration here. Event 1: When a UPA master port such as post 630 has a read-from-memory transaction ready, and the master counter is not at its allowed maximum (see box 430 in FIGS. 3A-3B), the read transaction is issued on the UPA.. Addressbus from UPA port 630 to the system controller 650. This is indicate by the event 1 (P..

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	REQ) along the UPA.. Addressbus 660 in FIG. 4, with the direction of the information indicated by the arrow, i.e. from the port to the SC. Event 2: The memory cycle i.e. RAS (read-address-strobe)/CAS (column-address-strobe) request issuance! is issued over memory control bus 670 to the memory banks 610 . . . 620. See vent 2 ("RAS/CAS") along bus 670. Event 3: The datapath is scheduled by a signal along the datapath control bus 680, and data items are accordingly delivered from memory to the datapath crossbar 625 via a memory databus 690 and UPA databus 700. This fulfills the read request.” Van Loo ’485 at col. 14, ll. 4–25. “Slave Read Requests: FIG. 6 FIG. 6 illustrates a read sequence to a slave device other than memory, and is similar to FIG. 2, but for this example a single master interface 710 and a single slave interface 720 are used, coupled by a system controller 720 and a datapath crossbar 730. Event 1 indicates the issuance of a read request P.. REQ on UPA address bus 740 to SC 720. In event 2, the SC 720 sends the P.. REQ on bus 750 to the slave interface 710. To do this, if there are several slave interfaces, the SC must first decode the address to ensure that the P.. REQ goes to the correct slave interface. Event 2 informs the slave interface to prepare the data to move through the datapath. When the data items are ready, then event 3 takes place, namely the sending of a P.. REPLY from the slave 710 to the SC 720 over bus 760. In event 4, a series of steps are executed to cause the master interface to receive the data: SC 720 schedules the datapath 730, and issues an S.. REPLY over bus 770 to the master interface 700. In addition, the SC issues the S.. REPLY over bus 780 to the slave 710, to drive the data, when it is ready, on the slave's UPA databus 790 via the datapath and over the databus 800 to the master interface 700.” Van Loo ’485 at col. 14, l. 49 – col. 15, l. 6.
14. The system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer.	Van Loo ’485 teaches “[t]he system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer,” including at least the following exemplary citations. <i>See above at row 2.</i>

Exhibit 596-17

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
15. The system of claim 13, wherein said system is either part of a storage area network (SAN) controller.	Van Loo '485 teaches "[t]he system of claim 13, wherein said system is either part of a storage area network (SAN) controller," including at least the following exemplary citations. <i>See above at row 3.</i>
16. The system of claim 13, wherein said system is part of a network attached storage (NAS) controller.	Van Loo '485 teaches "[t]he system of claim 13, wherein said system is part of a network attached storage (NAS) controller," including at least the following exemplary citations. <i>See above at row 4.</i>
17. The system of claim 13, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	Van Loo '485 teaches "[t]he system of claim 13, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element," including at least the following exemplary citations. <i>See above at row 5.</i>
18. The system of claim 13, wherein at least one third processing element executes said command and notifies said output queue to issue said response.	Van Loo '485 teaches "[t]he system of claim 13, wherein at least one third processing element executes said command and notifies said output queue to issue said response," including at least the following exemplary citations. <i>See above at row 6.</i>