

Exhibit 596-07

Invalidity of the '596 Patent by Goyal '579 and Other Prior Art

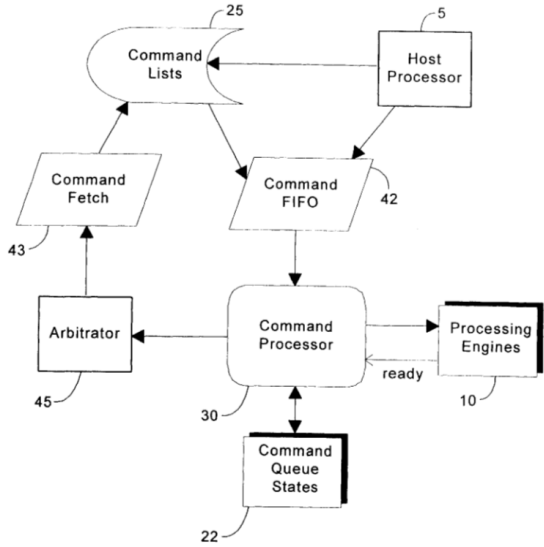
The asserted claims of U.S. Patent No. 8,924,596 are anticipated and/or rendered obvious in view of US6055579A (“Goyal '579”) alone or in combination with knowledge of one of skill in the art or other prior art.

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 Goyal '579. 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:	<p>To the extent the preamble is considered a claim limitation, Goyal '579 teaches “[a] system for dividing and synchronizing a processing task across a plurality of processing elements,” including at least the following exemplary citations.</p> <p>For example, Goyal '579 teaches a system for dividing and synchronizing processing tasks by disclosing, as non-limiting examples, a data processing system that includes processing engines, a host processor, and a command processor that divides and synchronizes tasks. <i>See, e.g.</i>:</p> <p>“A system for synchronization of data processing in a data processing system including multiple command queues is disclosed. The disclosed data processing system includes one or more processing engines associated with one or more command queues. The use of multiple command queues supports multiple priority levels, such that commands in higher priority queues may preempt commands in lower priority queues. Data processing is synchronized by queue commands that allow a processing engine to queue commands on the command queue of any processing engine in the data processing system, including its own. Multiple data dependencies are resolved by conditional queue commands and event counters that queue a command only when all of the conditions precedent to execution of a particular data processing command are satisfied. The hardware queuing of the disclosed invention advantageously synchronizes data processing with minimal software supervision and with minimal latency.” Goyal '579 at Abstract.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 961 844">FIG. 3</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

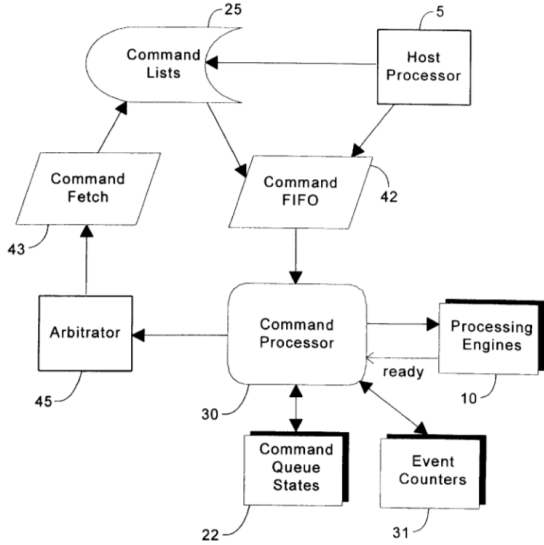
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“Data processing by multiple processing engines or processors such as encountered in, for example, video data processing, (e.g., multipass compositing), requires synchronization of the multiple processing engines. If the processing engines are not operating at the same speed (i.e., are not in lock step), then parallel processing generally, and video data processing in particular, requires extensive software overhead to coordinate processing activities and/or maintain data integrity.</p> <p>For example, video data processing typically entails sending the video data being processed through multiple processing engines, particularly during multipass compositing. Separate data streams may be simultaneously processed by distinct processing engines and effective coordination of the different processing engines is required to prevent processing engines from idling and undesirably slowing the total rate of data processing throughput.” Goyal '579 at col. 1, ll. 16–32.</p> <p>“In view of the foregoing, there is a need for a flexible system that synchronizes processing of data for multiple processing engines (or multiple processing queues for a single processing engine) in a data processing system such that processing time is optimized and processing engines are fully utilized with minimal software overhead and latency. By independently queuing commands to each processing engine, providing processing engines with the ability to queue commands without software intervention and by</p>

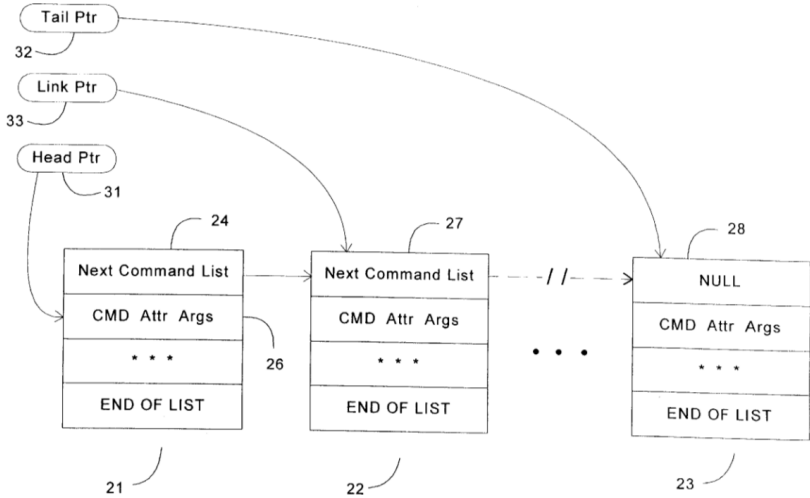
Exhibit 596-07

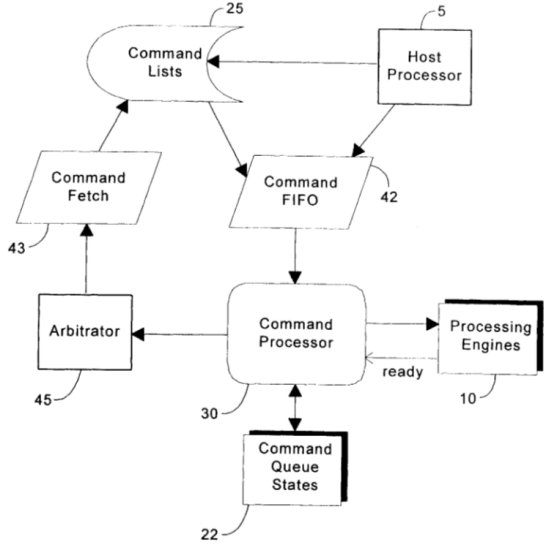
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>providing a mechanism by which individual commands may be conditionally queued, data dependencies between processing engines and within data streams can be fully resolved without requiring incorporation of complex data coherency systems.” Goyal ’579 at col. 2, l. 60 – col. 3, l. 6.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal ’579 at col. 3, ll. 7–14.</p> <p>“Accordingly, a data processing system with multiple processing engines can be readily implemented wherein the processing activities of the processing engines are efficiently coordinated for optimal data throughput with minimum idling of processing engines, software processing overhead and latency. Multiple data dependencies are readily resolved by the conditional queues and event counters disclosed herein. Efficient multiplexing of processing engines is further enhanced by the implementation of multiple command queues with varying priorities that provide a mechanism by which the present invention may be flexibly adapted to individual application requirements.” Goyal ’579 at col. 4, ll. 15–26.</p> <p>“A novel technique and apparatus for synchronizing data processing for a plurality of command queues for one or more processors or processing engines in a data processing system is described herein. For purposes of example only, the detailed description of the present invention is presented primarily with respect to a video processing subsystem. However, the present invention is not limited to video data processing and is equally applicable to any form of digital signal processing or data processing wherein a plurality of data processing engines are utilized. Moreover, the present invention is not limited to data processing subsystems but is also applicable to independent data processing systems.” Goyal ’579 at col. 4, ll. 47–58.</p> <p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.
[1a] an input queue implemented in hardware;	<p>Goyal ’579 teaches “an input queue implemented in hardware,” including at least the following exemplary citations.</p> <p>Plaintiffs have cited the use of an input queue implemented in memory through software as allegedly satisfying limitation [1a]. <i>See</i> Am. Compl. at ¶59; ’596 Infringement Contentions at 43–58. To the extent that Plaintiffs argue that Goyal ’579’s input queue is in memory Goyal ’579 teaches or renders obvious claim limitation [1a] under Plaintiffs’ apparent interpretation of the claim.</p> <p>For example, Goyal ’579 teaches an input queue implemented in hardware, by disclosing, as non-limiting examples, a command queue in a linked list. <i>See, e.g.:</i></p> <p>“A system for synchronization of data processing in a data processing system including multiple command queues is disclosed. The disclosed data processing system includes one or more processing engines associated with one or more command queues. The use of multiple command queues supports multiple priority levels, such that commands in higher priority queues may preempt commands in lower priority queues. Data processing is synchronized by queue commands that allow a processing engine to queue commands on the command queue of any processing engine in the data processing system, including its own. Multiple data dependencies are resolved by conditional queue commands and event counters that queue a command only when all of the conditions precedent to execution of a particular data processing command are satisfied. The hardware queuing of the disclosed invention advantageously synchronizes data processing with minimal software supervision and with minimal latency.” Goyal ’579 at Abstract.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 2</p> <p>Goyal '579 at Figure 2.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 961 844">FIG. 3</p> <p data-bbox="583 847 884 880">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

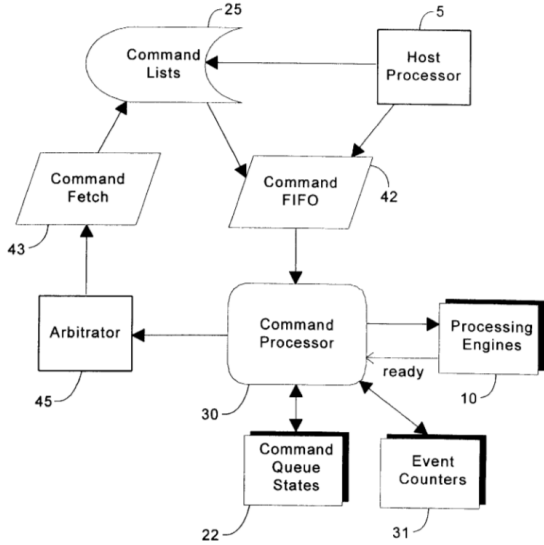
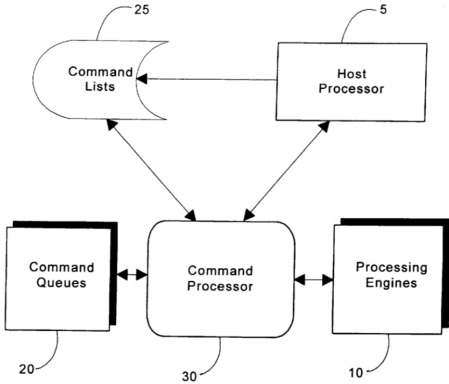
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The command processor monitors and maintains a set of queue pointers in an embodiment of the present invention where the command queues are implemented as linked lists which are stored in a shared memory in a data processing system which may be a subsystem in multipurpose computing system such as, for example, a video data processing system. The individual elements of the command queues may be actual commands, including the command arguments, or, as in a preferred embodiment, the elements of the command queue may be command tokens and pointers to the command arguments. By using command tokens, the command queues can be advantageously implemented in fast local memory which can be integrated with the data processing subsystem because the memory size does not have to be large enough to accommodate all pending commands and data to be processed.” Goyal '579 at col. 3, l. 60 – col. 4, l. 8.</p> <p>“The architecture of a data processing system in accordance with an embodiment of the present invention is illustrated in FIG. 1. The data processing system includes one or more processing engines 10 each of which is associated with one or more command queues 20. Each processing engine 10 can have a different number of associated command queues 20. For example, a first processing engine 10a may have one associated command queue 20a, a second processing engine 10b may have 3 associated command queues 20b-20d, and a third processing engine 10c may have two associated command queues 20e-20f. Alternatively, each</p>

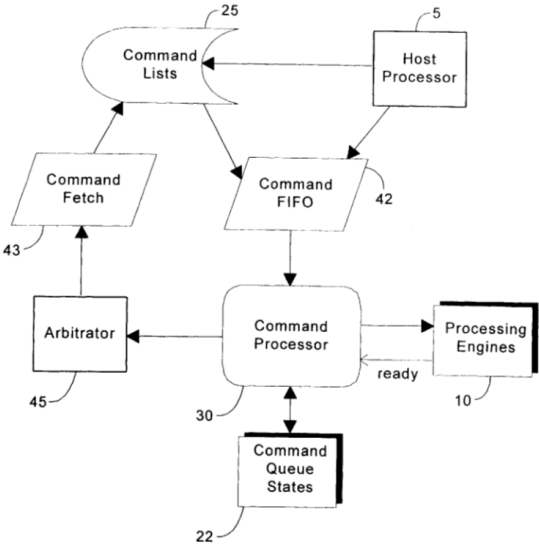
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>processing engine may be associated with the same number of command queues.” Goyal ’579 at col. 4, l. 59 – col. 5, l. 4.</p> <p>“Command processor 30 communicates with host processor 5, command lists 25, processing engines 10 and command queues 20. Command processor responds to instructions from host processor 5 by queuing command lists 25 on command queues 20. Command processor 30 also maintains the command queues 20, fetches commands from memory and deliver them to processing engines 10.” Goyal ’579 at col. 5, ll. 12–18.</p> <p>“In a preferred embodiment of the present invention, each command queue is maintained as a linked list data structure, specifically a singly linked list of command lists. Referring to FIG. 2, a command queue of a preferred embodiment is shown wherein command queue 20 is a singly linked list of command lists 21, 22, and 23. Each command list is a set of commands. A command list may have one or more commands, the size and composition of the command lists varying with the particular processing engine and various implementation specific system considerations. A command list may be constructed as any suitable data structure, including a linear queue or a linked list, although it is preferentially stored as a consecutive sequence of commands. Each command list may be stored in local memory or main memory.” Goyal ’579 at col. 6, ll. 1–15.</p> <p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal ’579 at col. 6, ll. 52–62.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p>
<p>[1b] an output queue implemented in hardware;</p>	<p>Goyal ’579 teaches “an output queue implemented in hardware,” including at least the following exemplary citations.</p> <p>Plaintiffs have cited the use of an output queue implemented in memory through software as allegedly satisfying limitation [1b]. <i>See</i> Am. Compl. at ¶59; ’596 Infringement Contentions at 58–72. To the extent that Plaintiffs argue that Goyal ’579’s output queue is in memory Goyal ’579 teaches or renders obvious claim limitation [1b] under Plaintiffs’ apparent interpretation of the claim.</p> <p>For example, Goyal ’579 teaches an output queue implemented in hardware, by disclosing, as non-limiting examples, a conditional queue. <i>See, e.g.:</i></p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p>
<p>[1c] a first processing element having access to said input queue and said output queue;</p>	<p>Goyal ’579 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations.</p> <p>For example, Goyal ’579 teaches a first processing element having access to said input queue and said output queue, by disclosing, as non-limiting examples, a host processor. The host processor has access to both the input queue and output queue. <i>See, e.g.:</i></p> <p>“A system for synchronization of data processing in a data processing system including multiple command queues is disclosed. The disclosed data processing system includes one or more processing engines associated</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>with one or more command queues. The use of multiple command queues supports multiple priority levels, such that commands in higher priority queues may preempt commands in lower priority queues. Data processing is synchronized by queue commands that allow a processing engine to queue commands on the command queue of any processing engine in the data processing system, including its own. Multiple data dependencies are resolved by conditional queue commands and event counters that queue a command only when all of the conditions precedent to execution of a particular data processing command are satisfied. The hardware queuing of the disclosed invention advantageously synchronizes data processing with minimal software supervision and with minimal latency.” Goyal ’579 at Abstract.</p>  <pre> graph TD HP[Host Processor 5] <--> CL((Command Lists 25)) CL <--> CQ[Command Queues 20] CL <--> CP[Command Processor 30] CQ <--> CP CP <--> PE[Processing Engines 10] </pre> <p>The diagram illustrates a data processing system architecture. At the top, a Host Processor (5) is connected via a bidirectional arrow to a Command Lists (25) component, which is represented by a semi-circular shape. Below the Command Lists, there are three main components: Command Queues (20), a Command Processor (30), and Processing Engines (10). The Command Lists (25) are connected to both the Command Queues (20) and the Command Processor (30) with bidirectional arrows. The Command Queues (20) and the Command Processor (30) are also connected with a bidirectional arrow. Finally, the Command Processor (30) is connected to the Processing Engines (10) with a bidirectional arrow.</p> <p>FIG. 1 Goyal ’579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 955 844">FIG. 3</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“Accordingly, a data processing system with multiple processing engines can be readily implemented wherein the processing activities of the processing engines are efficiently coordinated for optimal data throughput with minimum idling of processing engines, software processing overhead and latency. Multiple data dependencies are readily resolved by the conditional queues and event counters disclosed herein. Efficient multiplexing of processing engines is further enhanced by the implementation of multiple command queues with varying priorities that provide a mechanism by which the present invention may be flexibly adapted to individual application requirements.” Goyal '579 at col. 4, ll. 15–26.</p> <p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.</p>
[1d] at least one second processing element in communication with said first processing element;	Goyal ’579 teaches “at least one second processing element in communication with said first processing element,” including at least the following exemplary citations.

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>For example, Goyal '579 teaches at least one second processing element in communication with said first processing element, by disclosing, as non-limiting examples, a command processor. The command processor, the second processing element is in communication with the first processing element. <i>See, e.g.:</i></p> <pre>graph TD; 5[Host Processor] --> 25[Command Lists]; 5 --> 30[Command Processor]; 30 --> 20[Command Queues]; 30 --> 10[Processing Engines];</pre> <p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 3</p> <p>Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

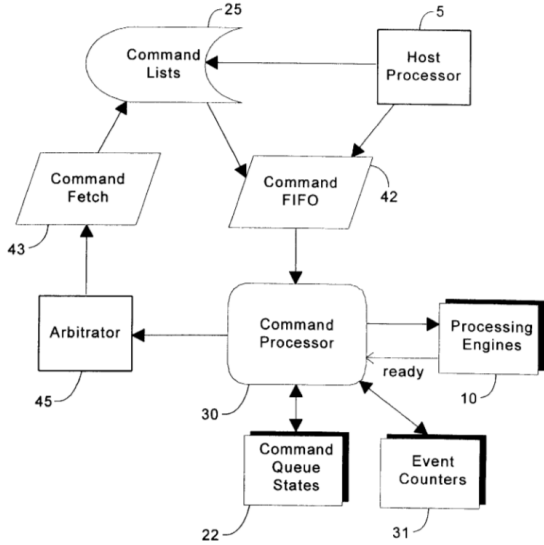
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The command processor monitors and maintains a set of queue pointers in an embodiment of the present invention where the command queues are implemented as linked lists which are stored in a shared memory in a data processing system which may be a subsystem in multipurpose computing system such as, for example, a video data processing system. The individual elements of the command queues may be actual commands, including the command arguments, or, as in a preferred embodiment, the elements of the command queue may be command tokens and pointers to the command arguments. By using command tokens, the command queues can be advantageously implemented in fast local memory which can be integrated with the data processing subsystem because the memory size does not have to be large enough to accommodate all pending commands and data to be processed.” Goyal '579 at col. 3, l. 60 – col. 4, l. 8.</p> <p>“Accordingly, a data processing system with multiple processing engines can be readily implemented wherein the processing activities of the processing engines are efficiently coordinated for optimal data throughput with minimum idling of processing engines, software processing overhead and latency. Multiple data dependencies are readily resolved by the conditional queues and event counters disclosed herein. Efficient multiplexing of processing engines is further enhanced by the implementation of multiple command queues with varying</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>priorities that provide a mechanism by which the present invention may be flexibly adapted to individual application requirements.” Goyal ’579 at col. 4, ll. 15–26.</p> <p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30. A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“Command processor 30 communicates with host processor 5, command lists 25, processing engines 10 and command queues 20. Command processor responds to instructions from host processor 5 by queuing command lists 25 on command queues 20. Command processor 30 also maintains the command queues 20, fetches commands from memory and deliver them to processing engines 10.” Goyal ’579 at col. 5, ll. 12–18.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p>

Exhibit 596-07

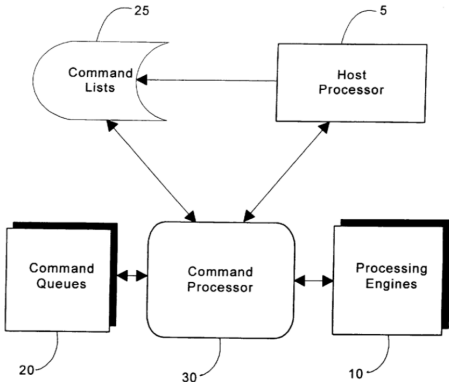
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.</p>
<p>[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;</p>	<p>Goyal ’579 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.</p> <p>For example, Goyal ’579 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 command queue state register that contains the head and tail pointer of a command queue. The command queue state register stores a value of indicative space and is accessible by both the first and second processing elements discussed above.</p> <p>Defendants have proposed a construction that “register” means “set of flip–flops that store a set of related bits.” Goyal ’579 teaches a reservation register that is a set of flip-flops that store a set of related bits. <i>See, e.g.</i>, Goyal ’579 at 6:58–62 (“In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.”). For example, the command queue state register 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> Goyal ’579 is a multiprocessor system that uses registers. A POSITA reading Goyal ’579 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>:</p>

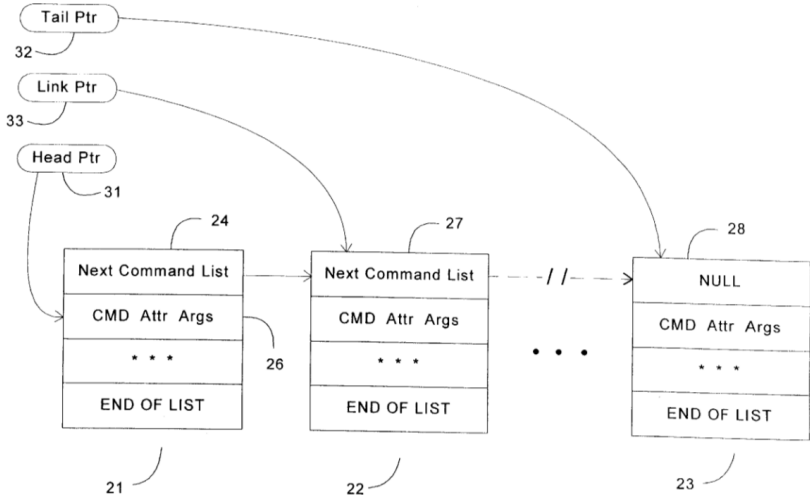
Exhibit 596-07

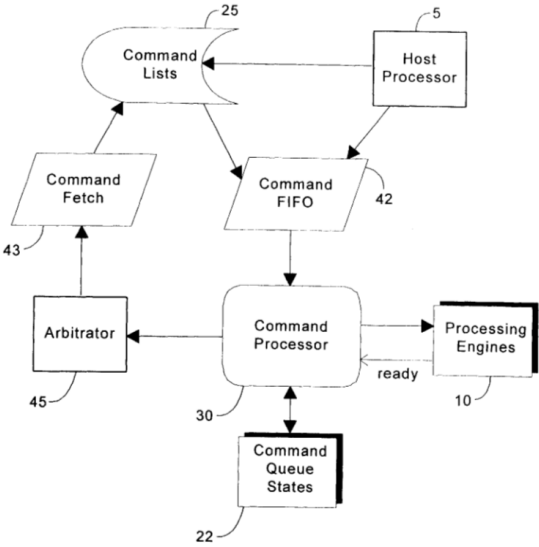
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal ’579 at col. 6, ll. 52–62.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p>
<p>[1f] computer storage storing instructions, which when executed by said at least one second processing element:</p>	<p>Goyal ’579 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations.</p> <p>For example, Goyal ’579 teaches computer storage storing instructions that are executed by said at least one second processing element, by disclosing, as non-limiting examples, a command processor that controls pointers based on instructions. The command processor executes those instructions stored in the computer storage. <i>See, e.g.:</i></p> <p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal ’579 at col. 6, ll. 52–62.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
<p>[1g] accesses said reservation register and reads said stored value;</p>	<p>Goyal '579 teaches “accesses said reservation register and reads said stored value,” including at least the following exemplary citations.</p> <p>For example, Goyal '579 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, a command processor that accesses the command queue state register. The command processor accesses and reads the stored value in the reservation register. <i>See, e.g.:</i></p> <p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal '579 at col. 6, ll. 52–62.</p>
<p>[1h] determines when said read value indicates available space in said input queue for said first processing element to issue a command;</p>	<p>Goyal '579 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.</p> <p>For example, Goyal '579 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, a command processor that reads the command queue state register. The command processor determines if the stored value indicates available space in the input queue for the first processing element to issue a command. <i>See, e.g.:</i></p> <p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal '579 at col. 6, ll. 52–62.</p>
<p>[1i] notifies said first processing element to issue said command to said input queue; and</p>	<p>Goyal '579 teaches “notifies said first processing element to issue said command to said input queue,” including at least the following exemplary citations.</p> <p>For example, Goyal '579 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</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>examples, the command processor examining the state of the command queue. The command processor notifies the first processing element to issue the command to the input queue. <i>See, e.g.:</i></p>  <p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="953 802 1083 846">FIG. 2</p> <p data-bbox="583 846 884 878">Goyal '579 at Figure 2.</p>

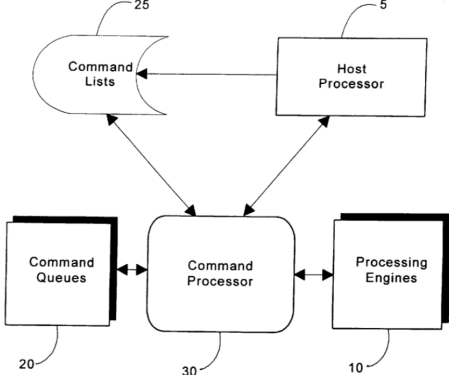
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 955 844">FIG. 3</p> <p data-bbox="583 847 882 880">Goyal '579 at Figure 3.</p>

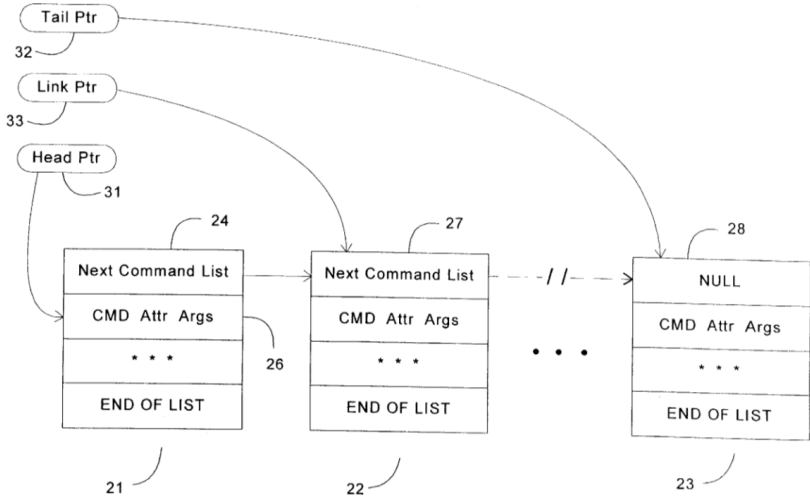
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

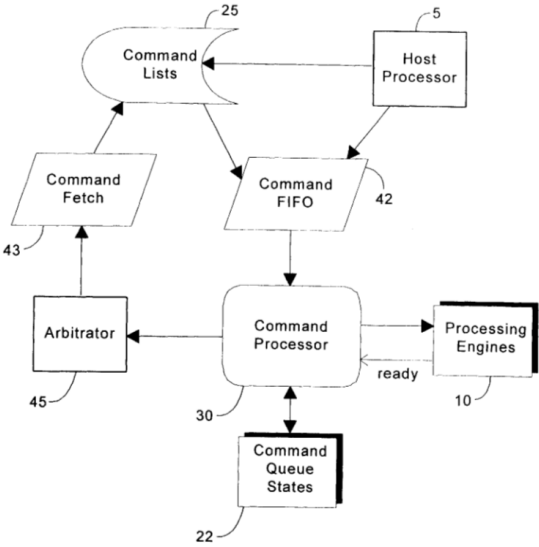
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“There are two categories of commands: queue management commands and processing commands. Processing commands may be specific to each processing engine (or type of processing engine) and are executed by a processing engine. Queue management commands are used to manage the command queues by, for example, adding commands to a processing engine command queue or initializing an event queue counter. One example of a queue management command is a QUEUE command that can queue a command list on a specific queue, either the queue in which the QUEUE command appears or on another queue which may be associated with another processing engine.” Goyal '579 at col. 3, ll. 33–44.</p> <p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
<p>[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.</p>	<p>Goyal '579 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.</p> <p>For example, Goyal '579 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, a host processor that receives the notification, issues the command, and receives a corresponding response from the conditional queue. <i>See, e.g.:</i></p>  <pre> graph TD HP[Host Processor 5] <--> CL[Command Lists 25] HP <--> CP[Command Processor 30] CL <--> CP CQ[Command Queues 20] <--> CP CP <--> PE[Processing Engines 10] </pre> <p style="text-align: center;">FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="953 802 1083 846">FIG. 2</p> <p data-bbox="583 846 884 878">Goyal '579 at Figure 2.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 955 844">FIG. 3</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

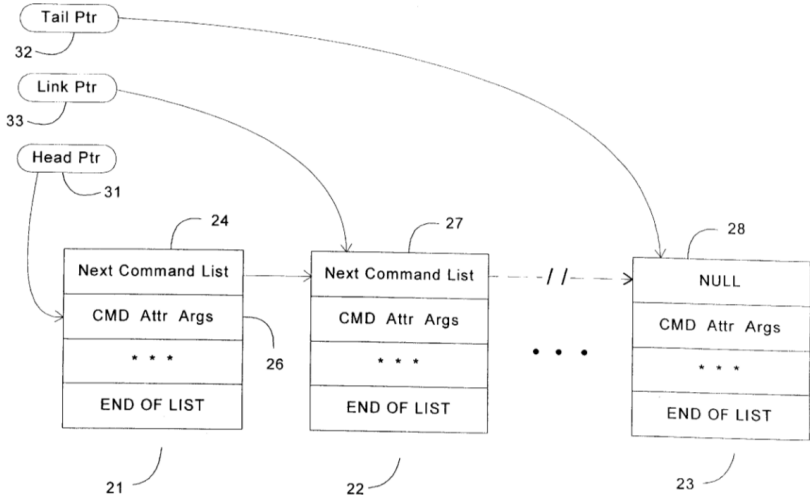
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“There are two categories of commands: queue management commands and processing commands. Processing commands may be specific to each processing engine (or type of processing engine) and are executed by a processing engine. Queue management commands are used to manage the command queues by, for example, adding commands to a processing engine command queue or initializing an event queue counter. One example of a queue management command is a QUEUE command that can queue a command list on a specific queue, either the queue in which the QUEUE command appears or on another queue which may be associated with another processing engine.” Goyal '579 at col. 3, ll. 33–44.</p> <p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30.</p>

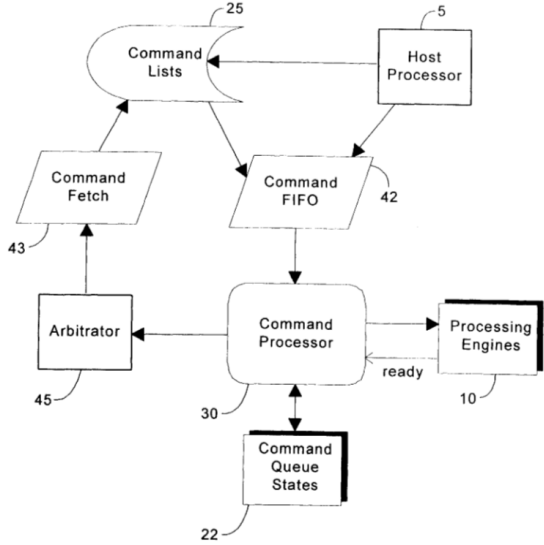
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p> <p>“7. The system of claim 6 further comprising:</p> <p>a conditional queue management command initialization establishing a condition precedent to queuing at least one of said commands in a command queue; and</p> <p>a conditional queue management command having a command name argument, a command queue argument, and a condition argument, said command name argument identifying at least one of said commands to be queued, said command queue argument identifying a command queue in which said at least one command is queued, and said condition argument identifying a condition precedent that must be met before at least one of said identified commands will be queued on said identified command queue.” Goyal ’579 at col. 11, ll. 34–47.</p>
<p>2. The system of claim 1, wherein said input queue is a first-in-first-out (FIFO) buffer.</p>	<p>Goyal ’579 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.</p> <p>For example, Goyal ’579 teaches wherein said input queue is a first-in-first-out (FIFO) buffer, by disclosing, as non-limiting examples, a FIFO data structure. <i>See, e.g.:</i></p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="953 802 1083 846">FIG. 2</p> <p data-bbox="583 846 884 878">Goyal '579 at Figure 2.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 961 844">FIG. 3</p> <p data-bbox="583 847 884 880">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“In an embodiment of the present invention, command processor 30 fetches the next command in each command queue and delivers it to the processing engine when the processing engine indicates that the processing engine is ready to execute a new command. In a preferred embodiment of the present invention, command processor 30 may prefetch commands before they are needed in order to ensure processing commands are available to processing engines 10 as soon as a processing engine can execute the command and therefore reduce latency.” Goyal '579 at col. 5, ll. 20–29.</p> <p>“In an alternative embodiment, the command queues for the processing engines may be implemented as first in first out (FIFO) data structures. The use of FIFO data structures simplifies maintenance of the command queues by the command processor. However, it limits the length of each command queue to the predetermined size of each FIFO queue whereas linked list command queues are advantageously flexible and only constrained by the limits of the system memory.” Goyal '579 at col. 7, ll. 38–46.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“In one embodiment of the present invention, the arbitrator 45 may be a simple FIFO queue. In an alternative embodiment, pending command requests may be arbitrated on the basis of priority (e.g., command requests from high priority queues selectively preempt command requests from low priority queues) and/or the processing engine on which the command will be executed. Therefore, the arbitrator may provide round-robin or weighted round-robin servicing of the processing engine requests. Once arbitrator 45 arbitrates between competing command requests, the command is fetched from command lists 25 by command fetch unit 43.” Goyal ’579 at col. 8, ll. 48–58.</p>
3. The system of claim 1, wherein said system is part of a storage area network (SAN) controller.	
4. The system of claim 1, wherein said system is part of a network attached storage (NAS) controller.	
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.	<p>Goyal ’579 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.</p> <p>For example, Goyal ’579 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 decrementing event counter. The event counter decrements the stored value by a predefined value. <i>See, e.g.:</i></p>

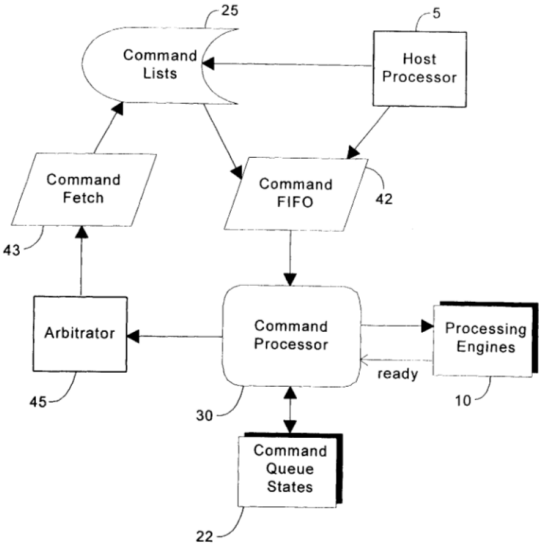
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“There are two categories of commands: queue management commands and processing commands. Processing commands may be specific to each processing engine (or type of processing engine) and are executed by a processing engine. Queue management commands are used to manage the command queues by, for example, adding commands to a processing engine command queue or initializing an event queue counter. One example of a queue management command is a QUEUE command that can queue a command list on a specific queue, either the queue in which the QUEUE command appears or on another queue which may be associated with another processing engine.” Goyal ’579 at col. 3, ll. 33–44.</p> <p>“Another example of a queue management command is a CONDQUEUE command used to conditionally queue a command list if the queue conditions are met, such as when the event counter associated with the CONDQUEUE command reaches a trigger value such as zero. The CONDQUEUE command may first modify the value stored in the event counter and then, if the modified value of the event counter equals a trigger value, queue a specified command list on a specified command queue. For example, the CONDQUEUE command may decrement the event counter and, if the event counter is zeroed, a specified command list would be queued on a specified command queue. The CONDQUEUE command therefore provides a mechanism for managing multiple dependencies amongst the processing engines.” Goyal ’579 at col. 3, ll. 45–59.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p>
<p>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.</p>	<p>Goyal ’579 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.</p> <p>For example, Goyal ’579 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, processing engines that execute commands. The processing engine executes the command and notifies the conditional queue to issue a response. <i>See, e.g.:</i></p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 955 844">FIG. 3</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“There are two categories of commands: queue management commands and processing commands. Processing commands may be specific to each processing engine (or type of processing engine) and are executed by a processing engine. Queue management commands are used to manage the command queues by, for example, adding commands to a processing engine command queue or initializing an event queue counter. One example of a queue management command is a QUEUE command that can queue a command list on a specific queue, either the queue in which the QUEUE command appears or on another queue which may be associated with another processing engine.” Goyal '579 at col. 3, ll. 33–44.</p> <p>“The architecture of a data processing system in accordance with an embodiment of the present invention is illustrated in FIG. 1. The data processing system includes one or more processing engines 10 each of which is associated with one or more command queues 20. Each processing engine 10 can have a different number of associated command queues 20. For example, a first processing engine 10a may have one associated command queue 20a, a second processing engine 10b may have 3 associated command queues 20b-20d, and a third processing engine 10c may have two associated command queues 20e-20f. Alternatively, each processing engine may be associated with the same number of command queues.” Goyal '579 at col. 4, l. 59 – col. 5, l. 4.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“Command processor 30 communicates with host processor 5, command lists 25, processing engines 10 and command queues 20. Command processor responds to instructions from host processor 5 by queuing command lists 25 on command queues 20. Command processor 30 also maintains the command queues 20, fetches commands from memory and deliver them to processing engines 10.” Goyal ’579 at col. 5, ll. 12–18.</p> <p>“In an embodiment of the present invention, command processor 30 fetches the next command in each command queue and delivers it to the processing engine when the processing engine indicates that the processing engine is ready to execute a new command. In a preferred embodiment of the present invention, command processor 30 may prefetch commands before they are needed in order to ensure processing commands are available to processing engines 10 as soon as a processing engine can execute the command and therefore reduce latency.” Goyal ’579 at col. 5, ll. 20–29.</p> <p>“In a preferred embodiment, each command in a command list consists of an operation to be executed by the processing engine (or the command processor, if the command is a queue management command) identified by a command token, command attributes, and command arguments. Command arguments are, for example, pointers that point to control or data information needed by the identified operation. The types of command arguments are dependent upon the data being processed, the command being executed, and the system architecture. Accordingly, a command argument may be, for example, a data value or it may be a direct or indirect reference to an input/output port or an image in memory.” Goyal ’579 at col. 6, ll. 30–42.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number</p>

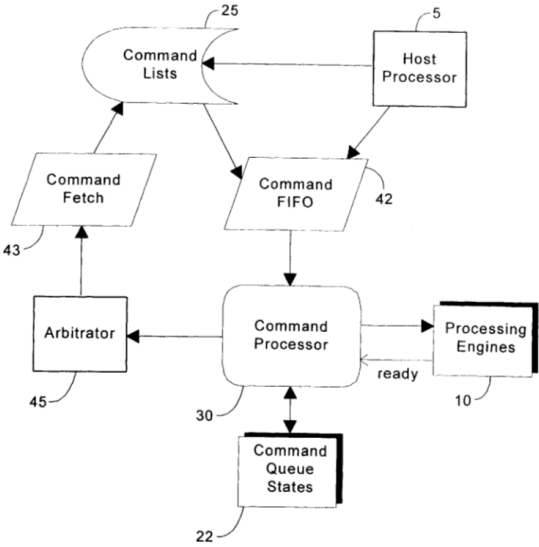
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p> <p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.</p> <p>“7. The system of claim 6 further comprising:</p> <p>a conditional queue management command initialization establishing a condition precedent to queuing at least one of said commands in a command queue; and</p> <p>a conditional queue management command having a command name argument, a command queue argument, and a condition argument, said command name argument identifying at least one of said commands to be queued, said command queue argument identifying a command queue in which said at least one command is queued, and said condition argument identifying a condition precedent that must be met before at least one of said identified commands will be queued on said identified command queue.” Goyal ’579 at col. 11, ll. 34–47.</p>
[7pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	<p>Goyal ’579 teaches “[a] system for dividing and synchronizing a processing task across a plurality of processing elements,” including at least the following exemplary citations.</p> <p><i>See</i> above at row [1pre].</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
<p>[7a] an input queue implemented in hardware;</p>	<p>Goyal '579 teaches “an input queue implemented in hardware,” including at least the following exemplary citations.</p> <p><i>See</i> above at row [1a].</p>
<p>[7b] an output queue implemented in hardware;</p>	<p>Goyal '579 teaches “an output queue implemented in hardware,” including at least the following exemplary citations.</p> <p><i>See</i> above at row [1b].</p>
<p>[7c] a first processing element having access to said input queue and said output queue;</p>	<p>Goyal '579 teaches “a first processing element having access to said input queue and said output queue,” including at least the following exemplary citations.</p> <p><i>See</i> above at row [1c].</p>
<p>[7d] at least one second processing element in communication with said input queue;</p>	<p>Goyal '579 teaches “at least one second processing element in communication with said input queue,” including at least the following exemplary citations.</p> <p>For example, Goyal '579 teaches at least one second processing element in communication with said input queue, by disclosing, as non-limiting examples, a command processor. The command processor, the second processing element is in communication with the input queue. <i>See, e.g.:</i></p> <p>“A system for synchronization of data processing in a data processing system including multiple command queues is disclosed. The disclosed data processing system includes one or more processing engines associated with one or more command queues. The use of multiple command queues supports multiple priority levels, such that commands in higher priority queues may preempt commands in lower priority queues. Data processing is synchronized by queue commands that allow a processing engine to queue commands on the command queue of any processing engine in the data processing system, including its own. Multiple data dependencies are resolved by conditional queue commands and event counters that queue a command only when all of the conditions precedent to execution of a particular data processing command are satisfied. The hardware queuing of the disclosed invention advantageously synchronizes data processing with minimal software supervision and with minimal latency.” Goyal '579 at Abstract.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 955 844">FIG. 3</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

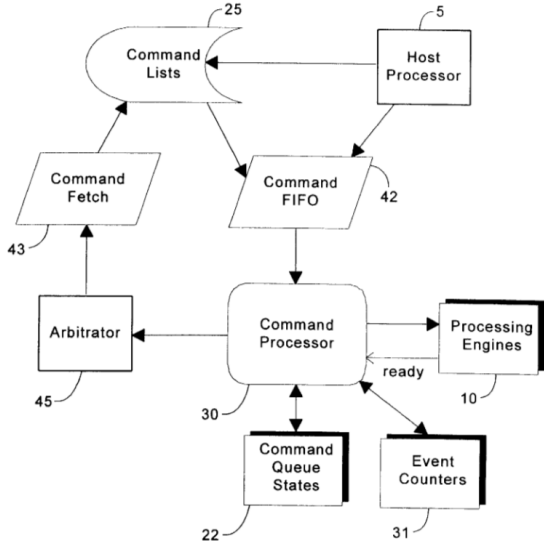
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“The command processor monitors and maintains a set of queue pointers in an embodiment of the present invention where the command queues are implemented as linked lists which are stored in a shared memory in a data processing system which may be a subsystem in multipurpose computing system such as, for example, a video data processing system. The individual elements of the command queues may be actual commands, including the command arguments, or, as in a preferred embodiment, the elements of the command queue may be command tokens and pointers to the command arguments. By using command tokens, the command queues can be advantageously implemented in fast local memory which can be integrated with the data processing subsystem because the memory size does not have to be large enough to accommodate all pending commands and data to be processed.” Goyal '579 at col. 3, l. 60 – col. 4, l. 8.</p>

Exhibit 596-07

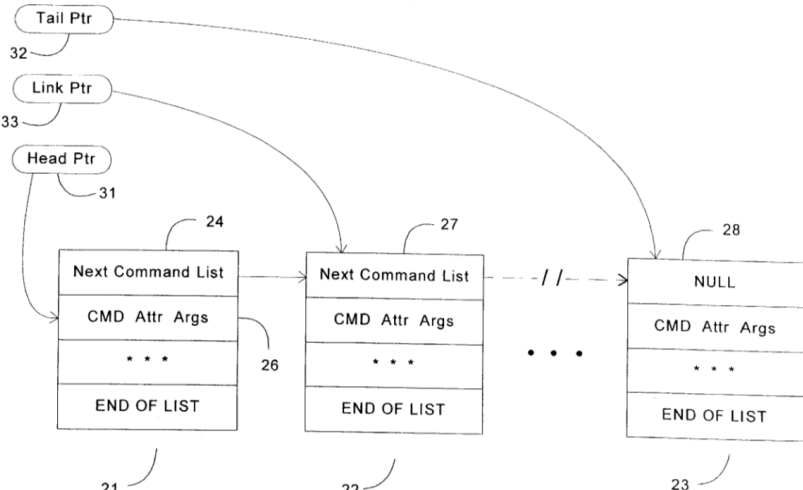
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30. A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“Command processor 30 communicates with host processor 5, command lists 25, processing engines 10 and command queues 20. Command processor responds to instructions from host processor 5 by queuing command lists 25 on command queues 20. Command processor 30 also maintains the command queues 20, fetches commands from memory and deliver them to processing engines 10.” Goyal ’579 at col. 5, ll. 12–18.</p> <p>“In an embodiment of the present invention, command processor 30 fetches the next command in each command queue and delivers it to the processing engine when the processing engine indicates that the processing engine is ready to execute a new command. In a preferred embodiment of the present invention, command processor 30 may prefetch commands before they are needed in order to ensure processing commands are available to processing engines 10 as soon as a processing engine can execute the command and therefore reduce latency.” Goyal ’579 at col. 5, ll. 20–29.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations</p>

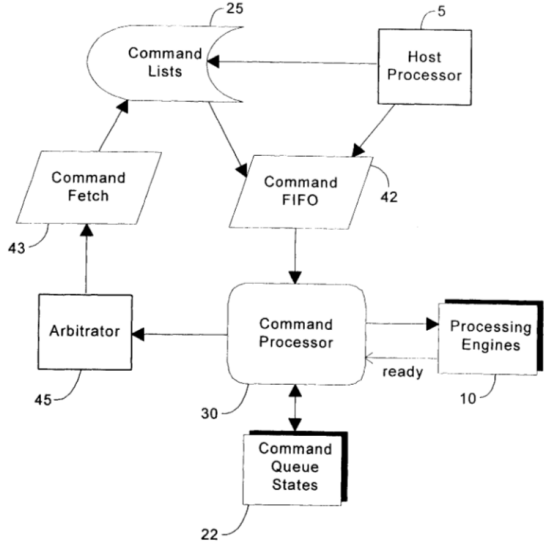
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p> <p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.</p>
<p>[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;</p>	<p>Goyal ’579 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.</p> <p><i>See above at row [1e].</i></p>
<p>[7f] computer storage storing instructions, which when executed by said at least one second processing element:</p>	<p>Goyal ’579 teaches computer storage storing instructions executed by said at least one second processing element, including at least the following exemplary citations.</p> <p><i>See above at row [1f].</i></p>
<p>[7g] receives a command from said first processing element;</p>	<p>Goyal ’579 teaches “receives a command from said first processing element,” including at least the following exemplary citations.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>For example, Goyal '579 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, a command processor that receives commands from a host processor. The command processor, the second processing element receives a command from the first processing element. <i>See, e.g.:</i></p>  <p style="text-align: center;">FIG. 2</p> <p>Goyal '579 at Figure 2.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 961 844">FIG. 3</p> <p data-bbox="583 847 884 880">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“Each command queue 20 is controlled by two pointers that are maintained by the command processor 20, a head pointer 31 and a tail pointer 32. Head pointer 31 points to the command 26 in the first command list in the command queue that is being currently executed or is the next command to be executed. Tail pointer 32 points to the last command list 28 in the command queue 20. In a preferred embodiment, head pointer 31 and tail pointer 32, (the command queue state), are maintained as register variables in command processor 30 to minimize command queue management processing time.” Goyal '579 at col. 6, ll. 52–62.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first</p>

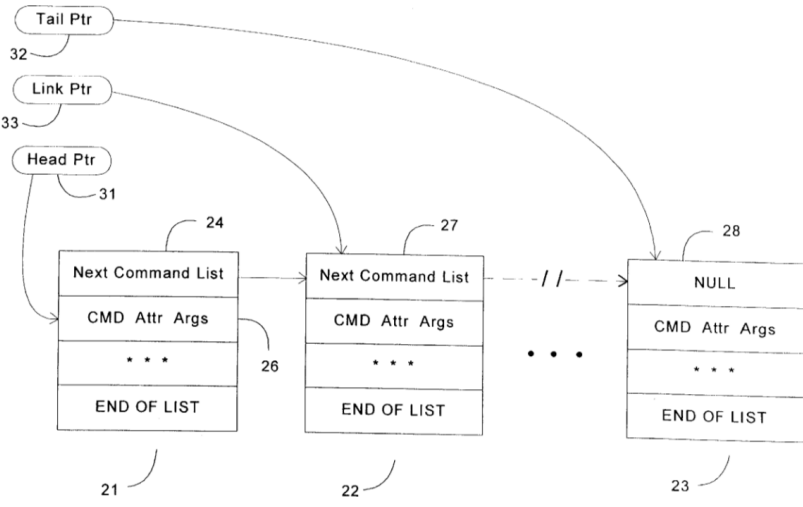
Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.
[7h] accesses said reservation register and reads said stored value;	Goyal ’579 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;	Goyal ’579 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	Goyal ’579 teaches “issues said command to said input queue on behalf of said first processing element,” including at least the following exemplary citations. For example, Goyal ’579 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, a command processor that issues commands on behalf of the host processor to the processing engines. The command processor 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
	<div data-bbox="583 240 1029 625"><pre>graph TD; HP[Host Processor 5] --> CL(Command Lists 25); HP --> CP(Command Processor 30); CL --> CP; CP --> CQ(Command Queues 20); CP --> PE(Processing Engines 10);</pre></div> <p data-bbox="583 803 882 885">FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="945 795 1092 844">FIG. 2</p> <p data-bbox="583 844 882 885">Goyal '579 at Figure 2.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 3</p> <p>Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“There are two categories of commands: queue management commands and processing commands. Processing commands may be specific to each processing engine (or type of processing engine) and are executed by a processing engine. Queue management commands are used to manage the command queues by, for example, adding commands to a processing engine command queue or initializing an event queue counter. One example of a queue management command is a QUEUE command that can queue a command list on a specific queue, either the queue in which the QUEUE command appears or on another queue which may be associated with another processing engine.” Goyal '579 at col. 3, ll. 33–44.</p> <p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“In an embodiment of the present invention, command processor 30 fetches the next command in each command queue and delivers it to the processing engine when the processing engine indicates that the processing engine is ready to execute a new command. In a preferred embodiment of the present invention, command processor 30 may prefetch commands before they are needed in order to ensure processing commands are available to processing engines 10 as soon as a processing engine can execute the command and therefore reduce latency.” Goyal ’579 at col. 5, ll. 20–29.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p>
<p>[7k] wherein said first processing element receives a response corresponding to said command from said output queue.</p>	<p>Goyal ’579 teaches “wherein said first processing element receives a response corresponding to said command from said output queue,” including at least the following exemplary citations.</p> <p>For example, Goyal ’579 teaches wherein said first processing element receives a response corresponding to said command from said output queue, by disclosing, as non-limiting examples, a host processor that receives the command result from the processing engine. The host processor receives a response from the conditional queue corresponding to the command. <i>See, e.g.:</i></p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 4 Goyal '579 at Figure 4.</p>

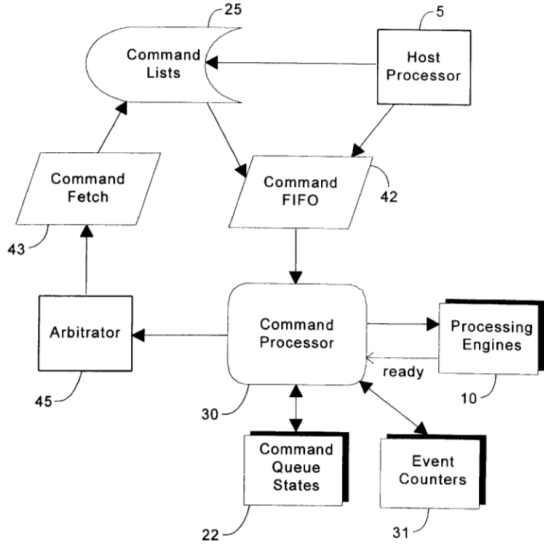
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p> <p>“7. The system of claim 6 further comprising:</p> <p>a conditional queue management command initialization establishing a condition precedent to queuing at least one of said commands in a command queue; and</p> <p>a conditional queue management command having a command name argument, a command queue argument, and a condition argument, said command name argument identifying at least one of said commands to be queued, said command queue argument identifying a command queue in which said at least one command is queued, and said condition argument identifying a condition precedent that must be met before at least one of said identified commands will be queued on said identified command queue.” Goyal ’579 at col. 11, ll. 34–47.</p>
<p>8. The system of claim 7, wherein said input queue is a first-in-first-out (FIFO) buffer.</p>	<p>Goyal ’579 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.</p> <p><i>See above at row 2.</i></p>
<p>9. The system of claim 7, wherein said system is part of a storage area network (SAN) controller.</p>	

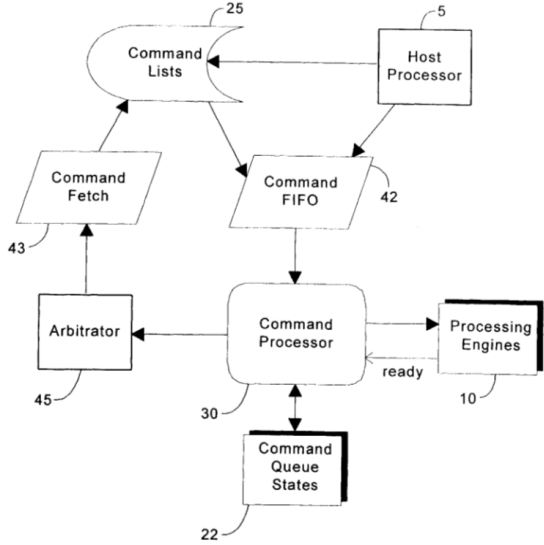
Exhibit 596-07

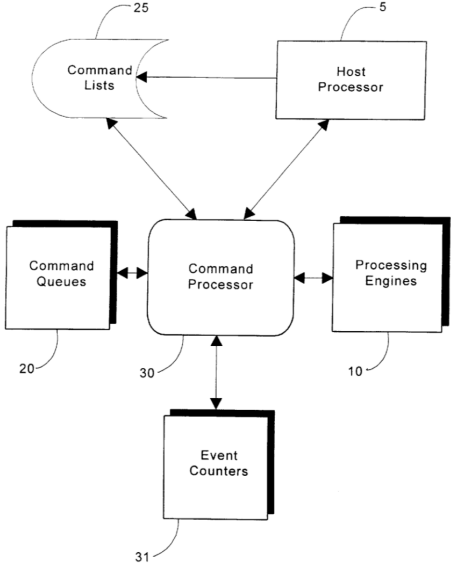
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
10. The system of claim 7, wherein said system is part of a network attached storage (NAS) controller.	
11. The system of claim 7, wherein said stored value is decremented by a predefined value prior to returning said stored value to said at least one second processing element.	<p>Goyal '579 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.</p> <p><i>See above at row 5.</i></p>
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.	<p>Goyal '579 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.</p> <p><i>See above at row 6.</i></p>
[13pre] A system for dividing and synchronizing a processing task across a plurality of processing elements comprising:	<p>Goyal '579 teaches "[a] system for dividing and synchronizing a processing task across a plurality of processing elements," including at least the following exemplary citations.</p> <p><i>See above at row [1pre].</i></p>
[13a] an input queue implemented in hardware;	<p>Goyal '579 teaches "an input queue implemented in hardware," including at least the following exemplary citations.</p> <p><i>See above at row [1a].</i></p>
[13b] an output queue implemented in hardware;	<p>Goyal '579 teaches "an output queue implemented in hardware," including at least the following exemplary citations.</p> <p><i>See above at row [1b].</i></p>
[13c] a first processing element having access to said input queue and said output queue;	<p>Goyal '579 teaches "a first processing element having access to said input queue and said output queue," including at least the following exemplary citations.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<i>See</i> above at row [1c].
[13d] at least one second processing element in communication with said input queue and said output queue;	<p>Goyal '579 teaches “at least one second processing element in communication with said input queue and said output queue,” including at least the following exemplary citations.</p> <p>For example, Goyal '579 teaches at least one second processing element in communication with said input queue and said output queue, by disclosing, as non-limiting examples, a command processor. The command processor, the second processing element is in communication with the input and output queue. <i>See, e.g.:</i></p> <p>“A system for synchronization of data processing in a data processing system including multiple command queues is disclosed. The disclosed data processing system includes one or more processing engines associated with one or more command queues. The use of multiple command queues supports multiple priority levels, such that commands in higher priority queues may preempt commands in lower priority queues. Data processing is synchronized by queue commands that allow a processing engine to queue commands on the command queue of any processing engine in the data processing system, including its own. Multiple data dependencies are resolved by conditional queue commands and event counters that queue a command only when all of the conditions precedent to execution of a particular data processing command are satisfied. The hardware queuing of the disclosed invention advantageously synchronizes data processing with minimal software supervision and with minimal latency.” Goyal '579 at Abstract.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>FIG. 1 Goyal '579 at Figure 1.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p data-bbox="829 803 961 844">FIG. 3</p> <p data-bbox="583 847 884 880">Goyal '579 at Figure 3.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The present invention can be readily implemented in any data processing system or subsystem with one or more processing engines. Each processing engine can queue commands on its own command queue(s) or on other processing engine command queues through a queue command. If centralized command control is desired, a command processor can control and maintain processing engine command queues.” Goyal '579 at col. 3, ll. 7–14.</p> <p>“The command processor monitors and maintains a set of queue pointers in an embodiment of the present invention where the command queues are implemented as linked lists which are stored in a shared memory in a data processing system which may be a subsystem in multipurpose computing system such as, for example, a video data processing system. The individual elements of the command queues may be actual commands, including the command arguments, or, as in a preferred embodiment, the elements of the command queue may be command tokens and pointers to the command arguments. By using command tokens, the command queues can be advantageously implemented in fast local memory which can be integrated with the data processing subsystem because the memory size does not have to be large enough to accommodate all pending commands and data to be processed.” Goyal '579 at col. 3, l. 60 – col. 4, l. 8.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“The data processing system of FIG. 1 also includes a host processor 5. Host processor 5 builds command lists 25 in memory and queues the command lists 25 in command queues 20 through command processor 30. A command list is a set of processing commands executed by a processing engine 10 and/or queue management commands executed by command processor 30.” Goyal ’579 at col. 5, ll. 5–11.</p> <p>“Command processor 30 communicates with host processor 5, command lists 25, processing engines 10 and command queues 20. Command processor responds to instructions from host processor 5 by queuing command lists 25 on command queues 20. Command processor 30 also maintains the command queues 20, fetches commands from memory and deliver them to processing engines 10.” Goyal ’579 at col. 5, ll. 12–18.</p> <p>“In an embodiment of the present invention, command processor 30 fetches the next command in each command queue and delivers it to the processing engine when the processing engine indicates that the processing engine is ready to execute a new command. In a preferred embodiment of the present invention, command processor 30 may prefetch commands before they are needed in order to ensure processing commands are available to processing engines 10 as soon as a processing engine can execute the command and therefore reduce latency.” Goyal ’579 at col. 5, ll. 20–29.</p> <p>“When the data processing system is initialized, head pointer 31 and tail pointer 32 are set to null, indicating that the command queue is empty. When host processor 5 creates a command list and instructs command processor 30 to add the command list to an empty command queue, the command list is added to the command queue by updating tail pointer 32 to point to the first field in the command list (which is the first and last command list in the queue at that time). Head pointer 31 is updated to point to the first command in the command list. In a preferred embodiment of the present invention where head pointer 31 and tail pointer 32 are register variables in command processor 30, the corresponding registers are loaded with the addresses of the first field in the command list and the first command in the command list (i.e., the second entry in the command list as the first entry is reserved for a pointer to the next command list in the queue). If a command list is the only command list or the last command list in a command queue, the first entry is null.” Goyal ’579 at col. 6, l. 63 – col. 7, l. 14.</p> <p>“Processing of data by various processing engines is advantageously accomplished in the present invention using QUEUE commands. A QUEUE command instructs the system to queue a command list identified in the QUEUE command arguments onto a command queue which identified in the queue command arguments. In a preferred embodiment of the present invention, the QUEUE command has two arguments: a pointer to the command list to be queued and a pointer to the command queue onto which the command list will be queued. By allowing one processing engine to queue commands for other processing engines, the operations</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>of the processing engines can be optimally synchronized to minimize latency and idle processing engine cycles. Moreover, software overhead is advantageously minimized and the need for interrupt processing to initiate execution of sequential dependent data processing in multiple processing engines is obviated. Accordingly, the operation of the processing engines is effectively self-coordinated by hardware queuing capable of rapidly responding to changes in the states of the processing engines.” Goyal ’579 at col. 7, ll. 47–66.</p> <p>“Referring to FIG. 3, operation of the command processor 30 is illustrated diagrammatically. When host processor 5 (or a processing engine 10) generates a new command list 25 and instructs command processor 30 to add the new command list to a command queue or when a processing engine 10 generates a ready signal indicating that the processing engine 10 is ready for the next command, command processor 30 examines the state 22 of each command queue associated with the processing engine 10. When a new command list is added, a determination is made as to whether the new command list is being added to the highest priority command queue, and, if so, the first command in the new command list is processed by command processor 30. If a processing engine 10 generated a ready signal, then command processor 30 processes the next command in the first command list in the highest priority command queue associated with processing engine 10.” Goyal ’579 at col. 7, l. 67 – col. 8, l. 15.</p> <p>“Command processor 30 processes a processing command (as opposed to a queue management command) by examining the command queue state 22, particularly head pointer 31 which points to the command to be fetched for processing engine 10. Command processor 30 generates a command fetch request which is communicated to arbitrator 45. Arbitrator 45 arbitrates amongst concurrently pending command fetch requests and communicates a selected command fetch request to command fetch unit 43. Command fetch unit 43 fetches the command pointed to by head pointer 31 from command lists 25 and queues the fetched command on command FIFO queue 42. When the fetched command reaches the head of command FIFO queue 42, command processor 30 dispatches the command to processing engine 10. Command processor 30 then advances head pointer 31 to the next command in the command list. Whenever head pointer 31 points to the last command in a command list, indicated by either an END OF LIST command or an end of list attribute of the last command in the list, head pointer 31 is reset to point to the first command in the next command list as identified by the first entry in the command list and/or link pointer 33.” Goyal ’579 at col. 8, ll. 16–37.</p> <p>“Queue management commands (i.e., QUEUE) are handled differently from processing commands for processing engines. The command processor 30 processes queue management commands. For example, if a QUEUE command is encountered, the command processor places the command list identified by the QUEUE command into the command queue identified thereby, modifying the corresponding tail pointer 32 and first</p>

Exhibit 596-07

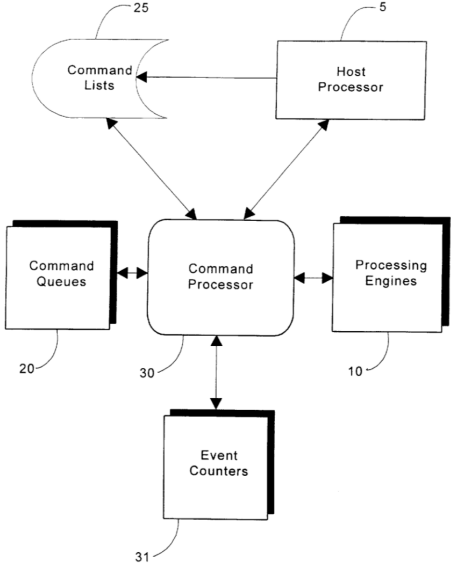
Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>entry of the last command list in the command queue to point to the newly added command list. Depending upon the state of the command queue, head pointer 31 and link pointer 33 may also be modified as discussed above.” Goyal ’579 at col. 8, l. 59 – col. 9, l. 3.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>“6. A data processing system comprising:</p> <p>a host processor;</p> <p>a plurality of processing engines coupled to said host processor;</p> <p>a plurality of command queues capable of receiving, storing and retrieving a plurality of commands, said commands including both processing commands and queue management commands, each of said command queues being associated with one of said plurality of processor engines; and</p> <p>a command processor executing said queue management commands, each of which has at least a command name argument and a command queue argument, said command name argument identifying at least one of said commands to be queued and said command queue argument identifying a command queue in which said at least one command is to be queued.” Goyal ’579 at col. 11, ll. 16–33.</p>
<p>[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;</p>	<p>Goyal ’579 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.</p> <p><i>See above at row [1e].</i></p>
<p>[13f] computer storage storing instructions, which when executed by said at least one second processing element:</p>	<p>Goyal ’579 teaches computer storage storing instructions that are executed by said at least one second processing element including at least the following exemplary citations.</p> <p><i>See above at row [1f].</i></p>
<p>[13g] receives a command from said first processing element;</p>	<p>Goyal ’579 teaches “receives a command from said first processing element,” including at least the following exemplary citations.</p> <p><i>See above at row [7g].</i></p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
[13h] accesses said reservation register and reads said stored value;	Goyal '579 teaches “accesses said reservation register and reads said stored value,” including at least the following exemplary citations. <i>See</i> above at row [1g].
[13i] determines when said read value indicates available space in said input queue to issue said command;	Goyal '579 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].
[13j] issues said command to said input queue on behalf of said first processing element; and	Goyal '579 teaches “issues said command to said input queue on behalf of said first processing element,” including at least the following exemplary citations. <i>See</i> above at row [7j].
[13k] receives a response corresponding to said command from said output queue.	Goyal '579 teaches “receives a response corresponding to said command from said output queue,” including at least the following exemplary citations. For example, Goyal '579 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, a command processor that receives a response from the conditional queue. The command processor receives a corresponding response from the conditional queue. <i>See, e.g.:</i>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	 <p>FIG. 4 Goyal '579 at Figure 4.</p>

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p style="text-align: center;">FIG. 5</p> <p>Goyal '579 at Figure 5.</p> <p>“The conditional queue command is similar to the queue command but also includes an additional argument relating to the condition or predicate which must be met before the command list (or command) identified by the conditional queue command is queued on the command queue identified therein. In other words, the identified command list (or command) is queued when the identified condition is satisfied.</p> <p>In a preferred embodiment, the condition or predicate is represented by an event counter. The event counter is initialized to a specific value representing the number of preceding events that must occur before the identified command list (or command) is queued on the identified command queue. A separate command initializes the event counter, and the additional conditional queue command argument is a reference to the event counter. Each time the conditional queue command is processed, the state of the referenced event counter is altered and examined.</p> <p>In one preferred embodiment, the event counter is initialized to a positive integer value and decremented each time a conditional queue command referencing the event counter is processed. If the event counter is decremented to zero, the command list referenced in the conditional queue command arguments is queued on the command queue referenced in the conditional queue command arguments. A zero value in the referenced</p>

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
	<p>event counter indicates that the identified condition has been satisfied. Accordingly, the identified command list is queued when the identified condition is satisfied.” Goyal ’579 at col. 9, ll. 15–44.</p> <p>“In a preferred embodiment, a conditional queue may be set up (i.e., associated with an event counter initialized to a particular value) by either host processor 5 or an appropriate queue management command processed by command processor 30. Host processor 5 may set up the conditional queue when a command list is set up, instructing command processor 30 to initialize a specified event counter to a specified value. Alternatively, command processor 30 may initialize an event counter when a SET EVENT COUNTER command (wherein the event counter and initial value are identified by the command arguments) is encountered in a command queue. The event counter 31 is initialized to a value corresponding to the number of precedent events which must occur before the identified command list is queued on the identified command queue. Thereafter, the event counter is decremented every time a CONDQUEUE command associated with the event counter is processed by command processor 30. When the associated event counter reaches zero, the identified command list is queued on the identified command queue.” Goyal ’579 at col. 9, l. 57 – col. 10, l. 9.</p> <p>“7. The system of claim 6 further comprising:</p> <p>a conditional queue management command initialization establishing a condition precedent to queuing at least one of said commands in a command queue; and</p> <p>a conditional queue management command having a command name argument, a command queue argument, and a condition argument, said command name argument identifying at least one of said commands to be queued, said command queue argument identifying a command queue in which said at least one command is queued, and said condition argument identifying a condition precedent that must be met before at least one of said identified commands will be queued on said identified command queue.” Goyal ’579 at col. 11, ll. 34–47.</p>
<p>14. The system of claim 13, wherein said input queue is a first-in-first-out (FIFO) buffer.</p>	<p>Goyal ’579 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.</p> <p><i>See above at row 2.</i></p>
<p>15. The system of claim 13, wherein said system is either part</p>	

Exhibit 596-07

Claim Language	Exemplary Evidence of Anticipation and/or Obviousness
of a storage area network (SAN) controller.	
16. The system of claim 13, wherein said system is part of a network attached storage (NAS) controller.	
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.	Goyal '579 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.	Goyal '579 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>