Verasonics®
The leader in Research Ultrasound™

Application Note

Management of Receive Data and DMA Transfers

Ken Linkhart
Director, System Engineering, Verasonics Inc.

April 2017

Summary

This document provides an overview of the Vantage system's functionality with regard to acquisition and storage of RF receive data. While conceptually simple, this topic can be confusing since it involves multiple steps: live acquisition of per-channel RF data with temporary storage in memory within the hardware system; DMA transfer of that data from the hardware system to a receive data buffer defined in memory on the host computer; and access to that data for software processing functions. The system is designed to automatically manage all of these steps based on very high-level definitions provided in an acquisition "SetUp…" script, so the user does not have to manage the low-level details associated with memory allocation and synthesis of DMA transfer commands. However, to achieve optimal data acquisition throughput and memory utilization the user must be aware of the constraints and assumptions that are implicit in the Vantage system design. This document is intended to explain those constraints and provide some user guidelines on how to most effectively utilize the RF data acquisition capabilities of the Vantage system.

Note that this document applies only to the Vantage 3.2.2 and later software releases; some aspects of receive data memory allocation and DMA generation were changed significantly in 3.2.1 and 3.2.2. This document does not explain in detail the differences from the 3.0 and earlier releases.

Note also that this document applies only to scripts written to allow RF data acquisition from a Vantage hardware system. Most of the constraints to be discussed are based on the management of the temporary memory buffer in the hardware, and the DMA transfers from there to the host computer. None of those constraints apply to a "simulation only" script that will not be used with the physical hardware system.

Table of Contents

Document Change History

Date	Section(s)	Description	Author
April 25, 2017	All	Rev A: Initial document for 3.2.2 release	KL

1. Receive Data Buffer Definition

As explained in the Vantage Sequence Programming Manual, receive data buffers are defined in the Matlab workspace through the structure ***Resource.RcvBuffer***. An array of *RcvBuffer* structures can be defined, each specifying a unique receive buffer number with its own attributes. Within each buffer, the size is specified by the three fields *rowsPerFrame*, *colsPerFrame*, and *numFrames*. In a simulation-only script any desired value can be used for each of these three dimensions of the ***RcvData*** array, but for acquisition from the hardware system some hardware-specific constraints are applied as explained below:

- ***rowsPerFrame***: This is the total number of samples available in the *RcvData* array for each channel. Since the data type is always int16, two bytes of memory are required for each individual sample. The actual number of samples used will be the sum of all samples acquired by all Receive events used to make up the frame. The *rowsPerFrame* value must be equal to or greater than that number; if it is greater the unused samples at the end of each column will never be written to by a DMA transfer, and thus will remain in the zero state from when the buffer was created and initially filled with zeros.

- ***colsPerFrame***: Each column represents the receive data from an individual receive channel in the hardware system. The number of columns required in the receive buffer may be greater than the number of receive channels that are actively being used, because of a constraint in the DMA transfers used in the Vantage system software: Receive channels in the Vantage hardware are combined into "channel groups" of 32 channels each, and each channel group individually manages its own DMA to the host computer. Those DMA transfers always include all 32 channels in the channel group, and thus the full 32 columns must be included in the receive buffer even if only a few of those channels are being used. Typically the system software will automatically determine the number of active channels groups, based on the connector type of the UTA module being used. For example, consider a Vantage 256 system with two 260-pin connectors each supporting 128 channels. If a script using only one of those two connectors is active, only the four channel groups associated with that connector will be enabled and thus the *colsPerFrame* value must be 128. (The 128 channels of the hardware system that are not being used will be automatically disabled and will not be included in the DMA transfers). But if a script for the P4-2v is being used, this probe has only 64 elements and thus the *Apod* arrays for TX and Receive structures in the script will only have 64 entries, representing the 64 active elements. The receive buffer will still be 128 columns wide, however, because the system software automatically enables all channel groups associated with element signals available at the selected connector(s)- even if the script being run does not use all of those channels. The unused channels will be filled with zeros and will be ignored by the Recon processing function, but they must exist in the definition of the receive buffer (and will be included in the DMA transfers). Note that this example does not apply to the Vantage 64 system; a Vantage 64 with the UTA 260-Mux module will have 64 active receive channels and 64 columns in the receive data buffer.

- ***numFrames***: Within a receive data buffer, any desired number of identical frames can be defined. Typical acquisition scripts cycle through a loop of multiple frames, so when you freeze or exit the script the entire loop of stored RF receive data frames will be available for review or offline processing. When running with the hardware system, if *numFrames* is greater than one then it must be an even number. This is because for the temporary buffer in the hardware only a "ping-pong" buffer of two frames will be allocated: all odd frames will use "ping" and all even frames will use "pong". This allows pipelining of the DMA transfers: after acquisition of frame 1 is complete, the DMA transfer of frame 1 from the "ping" memory can be underway at the same time that acquisition of new RF data for frame 2 is writing to "pong". The ping-pong approach ensures that new frame acquisition simultaneous with DMA from the previous frame will never collide, but only if the total number of frames is even (if the last frame in the buffer was odd, its DMA from "ping" would be underway at the same time that acquisition wrapped back to frame 1 and thus would also be writing to "ping"). To avoid this potential overwrite problem, the system enforces the even *numFrames* constraint: if your script defines a receive buffer with an odd number of frames greater than one, VSX will exit with an error condition. Defining and using a receive buffer with only one frame is allowed, but in this case it is up to the user to structure the timing in their script to prevent a new acquisition from writing to that frame before DMA transfer of the previous acquisition has been completed, and also to prevent DMA transfer of a new frame from starting until after software processing of the previous frame has been completed.

1.1 Mapping of Receive Events to Memory Within a Frame

Any desired number of acquisition events (each with a unique Receive structure) can be assembled to make up a frame of receive data for DMA transfer. The data within each column of the receive buffer will be allocated to individual receive events based on the number of samples acquired for that receive, and concatenated in the receive buffer in the order of the acqNum values for all receives within the frame. After running a script with VSX, you will be able to see the result of this mapping in the Receive structure from the Matlab workspace: new fields "startSample" and "endSample" will have been added to each Receive, representing explicitly the starting and ending row address within the RcvData buffer for the data from that Receive.

The actual number of samples acquired by a Receive event is specified by the startDepth and endDepth fields in the receive structure, representing the start and end points of RF data acquisition in units of wavelengths of Trans.frequency. The samplesPerWave field in the receive structure (which is also generated by the system software in response to the sample mode selected by the user) indicates exactly how many samples will be acquired per wavelength. Thus the total number of samples acquired by a receive event will be:

2 * samplesPerWave * (endDepth – startDepth)

The factor of two is for round-trip acquisition from a typical transmit-receive event. The system software then rounds up this value as needed so the resulting number of samples will be an integer number of blocks where each block is 128 samples. This quantization of the acquisition interval is a result of the code used to generate and manage the DMA transfers; the fixed block size has been set to allow maximum throughput in the DMA transfer rate. As an example, suppose you set startDepth to 2 and endDepth to 12, and samplesPerWave was precisely 4. This would require 80 samples, and thus the system would round endDepth up to 18 resulting in one block of 128 samples.

Note that any arbitrary sequence of Receive structures can be assembled to make up a frame- they do not need to be identical. For example you could interleave within the frame acquisition events for several different imaging modes, each potentially using different sample rates and acquisition intervals. The system simply takes each Receive in the order defined by the acqNum's, determines the number of blocks needed for that Receive, and concatenates them onto the columns of the receive buffer frame immediately after the last sample from the previous Receive acqNum.

2. Defining Receive Structures, Acquisition Events, and DMA Transfers

The typical sequence of structure definitions within a user "SetUp…" script (as illustrated by most of the Verasonics Example Scripts) is as follows:

1. Define the Receive Buffer(s)
2. Define the array of *Receive* structures to be used in the script
3. Define the Event Sequence (the array of *Event* structures that will be executed by the system).
4. Insert *TranferToHost* commands in the Event Sequence, at the points where the system is to initiate a DMA transfer of a frame of receive data from the hardware system to the host computer.

When the system initialization functions (within VSX and runAcq) process a user script to prepare for execution, a similar order is used. Each step in this process is explained below, to give you some insight into the assumptions the system makes that result in constraints on how you define your *Receive* and *Event* structures.

2.1 Receive Structure Definition

The mapping of receive data from each Receive structure to the associated memory locations within the receive data buffer is assigned based on the order in which the Receive structures have been defined within the 1 X N Receive structure array. It is important to recognize that this mapping is derived entirely from the Receive structure definitions themselves, without regard to how those Receive structures are used within the Event sequence. For example, if you defined a Receive structure that never was referenced in the

Event sequence, there would still be memory allocated for that Receive within the associated Receive Data buffer and frame. The system software requires that all Receive structures for a given buffer and frame within that buffer must be defined in a contiguous block within the Receive structure array, and the acqNum field for each mode 0 Receive must be assigned in increasing sequential order, starting with 1 for each frame. If you will be using mode 1 "accumulate" events within the frame, the mode 1 Receive structure for a given acqNum must be defined after the mode 0 Receive for that acqNum (You can meet this requirement be either defining each mode 1 Receive immediately after the associated mode 0, or you can define all of the mode 0 Receives for a frame in a sequential block and then go back and define the associated mode 1 Receives for that frame in another block. But in either case you must define all of the Receives for a given frame before going on to define the Receives for the next frame.).

Another critical requirement for receive data buffers with multiple frames is that a separate set of Receive structures must be defined for each frame, but the Receives within each frame must be identical except for the Receive.framenum field. This requirement is implied by the system's ability to use a "-1" frame pointer in both Recon processing and external processing functions operating on the receive buffer, meaning the processing will operate on whichever frame was most recently acquired and thus from the processing function's perspective all frames within the buffer must have an identical structure.

2.2 Event Sequence Definition

After all of the Receive structures that will make up each frame (and buffer) have been defined you can then define a sequence of acquisition Event structures that will reference those Receives, to acquire the data for each frame. Within the event sequence, you are free to execute the Receives that make up a frame in any arbitrary order- they do not have be executed in the order they were defined. There are, however, additional constraints on how the frame-to-frame acquisitions are structured within the event sequence:

- Individual acquisition events for different frames or buffers cannot be interleaved; you must acquire all Receives for a given frame before going on to a different frame.
- After the last acquisition for a frame, you must launch the DMA transfer for that frame with a 'transferToHost' command before any acquisition events for any other buffer or frame number. If you violate this rule the previously acquired frame will never be transferred to the host computer. Note however that you can place the transferToHost command either in the Event that included the last Receive for that frame or in a separate Event that occurs later in the sequence, as long as it comes before any Receive acquisition for a different frame or buffer.
- If you are acquiring all frames for a given receive buffer without interleaving acquisitions to a different buffer, you must acquire the individual frames in sequential order from 1 to the last frame in the buffer. If you fail to do this, you are at risk of a new acquisition overwriting data from the previous frame before it is transferred to the host computer, due to the "ping-pong" allocation of frames to the temporary buffer memory in the hardware system.

- In addition to executing the acquisition events for a given frame in arbitrary order, you are also free to omit individual Receives within the frame. The actual DMA transfer that is executed at the end of the acquisition events will only include the range from the smallest to the largest acqNum that was actually executed for that frame. For example suppose you defined a frame made up of five Receive events, with Receive acqNum's from 1 to 5. Then if the Event sequence only did three acquisitions of acqNum's 3, 5, 2 in that order followed by a transferToHost command, the actual DMA transfer would include the data in memory for frames 2, 3, 4, and 5. Thus in the receive data buffer in the host computer for that frame, the memory for Receive acqNum 1 would not be written to and would be filled with zeros from when the receive buffer was defined and initialized. The memory for Receive acqNum 4 will contain data from the associated locations in the temporary buffer in the hardware system but since no receive acquisition event has been executed for acqNum 4, this will be "garbage data" either from uninitialized memory in the hardware system or from a previous, unrelated acquisition event.

CAUTION: The Vantage software does not parse the event sequence thoroughly enough to detect all possible violations of the constraints listed above. In some cases, if you inadvertently violated one of these constraints (such as through an indexing error when defining or accessing the Receive structures) the system might appear to be running normally with no warning or error conditions reported, and yet the actual data that appears in the receive buffer may not be what was intended. After creating a new script or making extensive modifications to an existing script, it is wise to use the EventAnalysisTool to review the actual event sequence generated by the script to make sure there are no obvious indexing errors, and then test the script using a phantom or other test object to make sure the actual receive data acquisition and processing are functioning properly.

2.3 DMA Command Generation

The Vantage software will automatically create the DMA commands for you, to allow the DMA transfer of receive data from the temporary buffer in the hardware system to the receive data buffer in the host computer's memory. The only thing you need to do in the user setup script is insert 'transferToHost' sequence control commands in the Event sequence at the appropriate points: after all acquisition events for a specific receive data frame has been completed and before any acquisition events for a different receive data buffer or frame. Multiple frames or buffers cannot be combined in a single DMA. Listed here are guidelines and constraints you must conform to when adding 'transferToHost' commands to a SetUp script:

- A separate Sequence Control 'transferToHost' command must be created for each DMA transfer in the Event sequence. This is because each transferToHost will be converted into a unique DMA command with the source and destination memory addresses for a specific receive buffer and frame number. If you used the same

SeqControl index number for a transferToHost command at more than one point in the script, the DMA command generated for the last one would overwrite the DMA from the previous instance, resulting in the wrong data being written to a receive buffer location (and possibly a Matlab crash!).

- The system software determines which Receive structures will be included in a DMA by examining all acquisition events in the event sequence prior to the associated transferToHost command, back to either the previous transferToHost command or the beginning of the sequence. If this sequence of events includes Receive structures identifying more than one receive data buffer number or frame number, only the last buffer and frame will be included in the DMA and the previous acquisitions will never be transferred to the host computer. (This situation will not result in an error condition, since in some cases "dummy" acquisition events whose data will not be used are deliberately included in the event sequence).
- As explained in the Sequence Programming manual, conditions and arguments can be added to transferToHost commands to synchronize the processing functions running on the host computer with receive data acquisition and DMA from the hardware system. The use of these capabilities has no impact on the actual content of the individual DMA commands or the requirements for structuring the sequence of acquisition events and transferToHost commands, and so they are not discussed here.
- Note that when running a script in simulation mode, the DMA transfers are not exercised at all. The transmit-receive simulation software running on the host computer writes directly to the receive data buffer that will read from by Recon or other processing routines. Thus you may have bugs in the transferToHost commands that will prevent the script from running properly with the hardware system, and yet that same script will run perfectly in simulation!
- If the constraints given in the previous sections on Receive structure and Event sequence definition have not been followed, the result may be that DMA transfers will not be functioning properly even though the script appears to be running with no errors.

3. System Constraints on Receive Buffer Size

Generally speaking, both the Vantage software and the Matlab application impose no fixed maximum limits on size of a receive data buffer, or the number and duration of the individual Receive events used to fill a receive data frame. There are, however, practical and physical limits that will be encountered if you attempt to define an extremely large acquisition sequence and/or receive data buffer size. These limits are discussed in the subsections below, along with some tradeoffs you can make in your script to live within them.

3.1 Receive Buffer size in the Host Computer

There is no fixed limit on any of the three dimensions of a receive data buffer, or on the number of separate buffers that can be defined. The practical limit here is simply the amount of physical memory available on the host computer (or the size of the subset of that memory that has been allocated to the Matlab application). Note also that system memory is also being used by any processing functions that are active, temporary data buffers that may be created within a processing function, and other data buffers defined by the Vantage software (such as the ImageBuffer and InterBuffer that can be defined in a user SetUp script). Thus a script that only does RF data acquisition for offline processing may be able to user a much larger receive data buffer than a script doing extensive Recon and image data processing.

If the amount of memory needed exceeds what is physically available, the Operating System may automatically start moving some of the data out to 'virtual memory' stored on a disk drive. This may occur with no warning or error messages being generated, and the only symptom will be dramatically reduced acquisition and processing throughput. In some cases, the reduced throughput may in turn lead to timing problems within the Vantage software or the Matlab application, eventually resulting in a software crash. A very general "rule of thumb" is that the total size of all receive data buffers should be limited to roughly one fourth of the memory available on the host computer. If your application will exceed that limit, you should consider installing more memory in the computer, and/or breaking the acquisition into smaller chunks that can be streamed off to a disk drive or other large-capacity storage device with transfer rates adequate for your requirements.

3.2 Temporary Receive Data Buffer size in the Vantage Hardware System

As explained previously, all receive data acquisition events temporarily store their data in a memory buffer in the hardware system. After a full frame of data has been acquired, a DMA transfer is launched to move that frame up to the host computer. After the DMA transfer is complete, the same memory locations can be re-used to temporarily store a different acquisition frame. All Vantage hardware systems built to date have the same fixed memory space available, of 2 GBytes per channel group of 32 channels or equivalently 64 Mbytes per channel. This memory space is shared by three separate functions:

- Temporary storage of receive RF data; a separate buffer is allocated for each receive buffer defined in the user script but within each buffer only one or two frames are allocated even if more than two frames have been defined on the host computer.
- "Descriptor" memory space, for the descriptor tables used to program the operating state of the hardware for each individual transmit and receive event defined in the script (e.g. transmit waveforms, receive processing filter coefficients, etc.).
- "Scatter-gather lists" (SGL), the tables of source and destination memory addresses for each individual DMA transfer that has been defined in the script. For maximum data acquisition throughput, all of these tables are pre-calculated

when a script is initialized and then stored in the hardware system to eliminate any processing overhead when launching a DMA while the script is running.

The amount of memory needed for each of these three functions can vary widely, depending on the nature of the script. For example, a simple script with a small number of acquisition events each of which captures a very large block of receive data may be able to use nearly all of the 64 Mbytes per channel for receive data. At the other extreme a very complex script with thousands of TX and Receive structures may use most of the memory for descriptors, leaving only a small fraction of the 64 Mbytes for receive data. Starting with the Vantage 3.2 software releases the system uses the hardware memory as efficiently as possible by allocating only what is actually needed for each of the three segments. Thus the size of the individual segments is immaterial as long as the total over all three does not exceed 64 Mbytes (in 3.0 and all earlier Vantage releases a fixed allocation was used, of 32 Mbytes receive data, 16 Mbytes descriptors and 16 Mbytes SGL's).

3.3 Determining the Actual Hardware Memory Allocation

It is difficult to predict in advance what the channel group memory allocation will be by examining the SetUp script, but Verasonics provides a simple utility to determine the actual usage when the script is loaded for execution. If you add the following line to a SetUp script, the allocation of memory in the hardware system will be displayed at the Matlab command prompt:

```
Resource.VDAS.halDebugLevel = 1;
```

After running a script the allocation of memory within each channel group, as well as the separate sequence memory space usage, will be listed in the following values (displayed at the Matlab command prompt):

- **numberOfAcquisitionBlocks**: the total receive memory space in blocks; each block is 8192 bytes.
- **sizeDescriptorsTotal**: the total number of bytes allocated to descriptors
- **sizeSglsActuallyUsed**: the total number of bytes allocated to DMA scatter-gather lists
- **sizeSglsTotal**: the total amount of memory that was available for SGL use, after the receive data and descriptors were allocated.
- **sizeSequencerProgrammingTotal**: The total amount of memory available for storing the hardware event sequence programming data.
- **sizeSequencerProgrammingActuallyUsed**: The amount of memory actually being used by this script for storing the hardware event sequence programming data.

Since the total amount of memory available for each channel group is 2 GBytes (2^31, or 2,147,483,648 bytes), the sum

(numberOfAcquisitionBlocks * 8192 + sizeDescriptorsTotal + sizeSglsTotal)

should always equal that value. The amount of memory unused by the script will be **(sizeSglsTotal – sizeSglsActuallyUsed)**. To convert these values to the amount of memory used and available per channel, divide them by 32. Remember that the receive data is stored as 16 bit integers, so the number of receive samples will always be one half of receive buffer size in bytes.

Note that for a typical receive buffer definition using multiple frames in the Matlab workspace, the channel group memory buffer will only have two frames and thus will be much smaller than the memory space used for that buffer on the host computer. To minimize the receive data buffer size in channel group memory even further, there are two approaches you can use to force an allocation of only one frame instead of two:

1. If the Receive Data buffer definition in the script has only one frame, then only one frame will be allocated in channel group memory.
2. If "subframe" DMA transfers (see section 3.4 below) are being used within each frame, only one frame will be allocated in channel group memory regardless of the number of frames used in the script. This is because the multiple subframe DMA transfers provide the equivalent of the ping-pong allocation of two frames with only a single DMA per frame: acquisition of subframe 2 can proceed while DMA transfer of subframe 1 is underway. Thus the multiple subframe DMA's eliminate all risk of a read-write collision, so the ping-pong pair of frames is not needed.

Another constraint within the hardware system that may limit the size or complexity of your script is the sequence memory space, used to store the event sequence and associated sequence control commands. The sequence memory is associated with the ASC FPGA (Acquisition System Controller) on the backplane in the hardware system, and is physically separate from the channel group memories associated with each channel group on the acquisition modules. A script with a large number of events and sequence control commands is likely to run out of sequence memory before it has exhausted the available channel group memory. The value **sizeSequencerProgrammingActuallyUsed** tells you how much sequence memory the current script is actually using; the value **sizeSequencerProgrammingTotal** is the total amount of sequence memory available.

If you have a script that cannot run because it has exceeded one of the hardware system memory space limitations, you can use the Resource.VDAS.halDebugLevel = 1 command to determine which segment of memory is the bottleneck and use that information to guide you to the 'least painful' compromise to get the script to run.

3.4 DMA Transfer Maximum Size Limit: The SubFrame DMA Feature

The maximum allowed size of an individual DMA transfer is yet another constraint that may affect you when defining very large receive buffer frames. This limit comes from the host computer operating system, and is independent of the Vantage system design or the Matlab application. Typically this limit 2 GBytes, but it may vary depending on the version of OS you are using. Since the Verasonics system design requires you to DMA all the content of a receive buffer frame before acquiring another one, this 2 GByte per-DMA limit can be far more restrictive than the amount of memory installed on the host computer, or available in the channel groups of the hardware system (up to 16 GBytes if you are using all 256 channels of a Vantage 256 system).

In earlier Vantage software releases we had no way of avoiding this limit since the one-to-one mapping of receive data frames to DMA transfers is rigidly enforced by our overall system design. The only workaround was to divide the desired frame into smaller frames within a receive buffer (or in multiple buffers) and then re-assemble the desired frame on the host computer after all acquisitions and DMA transfers had been completed. Starting with the Vantage 3.2.2 release we now have a much simpler scheme to solve this problem: the "subframe DMA". The concept is to allow you to define arbitrarily large receive buffer frames and associated acquisition event sequences, but eliminate the requirement that the entire frame must be transferred with a single DMA command after all data for the frame has been acquired. Instead you can now insert multiple 'transferToHost' commands within the acquisition sequence of events for that frame; each of the associated DMA commands will only transfer the new data that has been acquired since the previous DMA. Any desired number of subframe transferToHost commands can be inserted within the acquisition events for the frame, at arbitrary points (they do not need to be uniformly spaced). There still must be a final transferToHost after the end of all acquisition events for the frame and before any acquisition to any other frame or buffer; this final DMA will be the one used to inform the software processing functions that acquisition and DMA transfer of the entire frame is complete. Note that the intent of the "subframe" DMA transfers is just to get around the maximum DMA size limit and to reduce memory usage in the channel group temporary RF data buffer; as far as Recon or other RF processing functions are concerned the subframe DMA's are invisible and the entire frame is presented to the processing at one point in time after the last subframe DMA for the frame has been completed.

To use the subframe DMA feature, the only change you need to make in a SetUp script is to add the additional transferToHost commands at the desired intermediate points in the acquisition event sequence for the frame. An example of this is included in the "…128RyLnsSubFrmDma" example scripts for the L7-4, L11-4v, and L11-5v probes in the ExampleScripts / Vantage 128 and 256 folder. These examples also illustrate the use of the Resource.VDAS.halDebugLevel = 1 command.

To ensure the subframe DMA feature will function properly in your script, it is critical you adhere to all of the guidelines listed in section 2 of this document for defining the Receive structures that will make up a frame, and the acquisition event sequence that will acquire the frame and initiate the DMA transfers. Some additional constraints specific to the subframe feature are listed here:

- Subframe acquisitions to other frames or buffer numbers cannot be interleaved in between the subframe acquisitions for a particular frame. All of the acquisition events and subframe transferToHost commands for a frame must be completed before an acquisition event for any other frame or buffer.
- The acquisition events that will be included in a subframe DMA can be executed in any arbitrary order in the event sequence, but they must make up a contiguous block of acqNum's since the subframe DMA command will include all acqNum's between the lowest and the highest values found for that DMA. For example, suppose you defined a frame made up of four Receive events (acqNum's 1, 2, 3, 4) but in your event sequence you acquired #1 and #4, then did a subframe DMA, then acquired #2 and #3 and then did the final DMA for the frame. In this case the first DMA would actually transfer data for all four acquisitions, adding meaningless overhead to transfer #2 and #3 since they will be transferred again by the final DMA of the frame after those acquisitions were completed.
- Note that the smallest subframe you could define would be the DMA transfer of a single acquisition event. This means you cannot define a single receive event that will acquire more than 2 GBytes of data since there will be no way to transfer it to the host computer without exceeding the maximum size limit for an individual DMA.
- The partition of a frame into two or more subframes can be done however desired, but it should be done the same way for all frames within the same buffer to ensure you will not have a read-write collision at any point in the acquisition and DMA sequence (Note that when subframe DMA transfers are being used, only one frame of memory is allocated in the channel group temporary buffer. Thus the last subframe DMA of frame N and the first subframe acquisition of frame N+1 may be underway at the same time. To ensure there will not be any read-write collisions, the acqNum's included in the last subframe of frame N must not overlap any of the acqNum's in the first subframe of frame N+1. Alternatively, you can ignore this constraint if the timing in your script ensures acquisition and DMA of a new frame will not start until after DMA and processing of the previous frame have been completed).
- If you are using synchronous handshaking between the hardware and software event sequences through the 'waitForProcessing' condition applied to the transferToHost commands, this must be done only for the final transferToHost command for each frame. The preceding subframe transferToHost commands must not have the waitForProcessing condition. In addition, the argument value associated with the waitForProcessing condition must reference the SeqControl index number of the final transferToHost of some other frame, not one of the subFrame DMA's. If you violate these constraints the script is likely to hang up until it either exits with a DMA timeout error or you force quit Matlab.
- Note that the Vantage system constraint that a new DMA transfer cannot start until after the previous DMA has finished applies to all DMA's, regardless of

whether they represent a subframe or an entire frame. Thus if you have broken a frame acquisition sequence into subframes and the acquisition time for a subframe is shorter than the DMA transfer time for the previously acquired subframe, the hardware acquisition sequence will be forced to pause at the next subframe boundary. This will not trigger an error condition (but may result in a "missed timeToNextAcq" warning message). In an acquisition sequence where PRI timing is critical (such as for Doppler detection), you must take care that the PRI for each event results in an average RF data acquisition rate that is well below the DMA transfer data rate.

Note that while the subframe DMA concept is valuable for reducing the size of individual DMA's to less than the max DMA size limit, you should avoid going too far to the other extreme. Since the execution of any DMA transfer includes a fixed overhead time of a few milliseconds in addition to the time required to actually transfer the data, breaking a frame up into a large number of small DMA's will reduce the net DMA data transfer rate that can be achieved. For optimal throughput, individual DMA's should be larger than 64 Mbytes but smaller than the 2 GByte maximum size limit (see section 4, "DMA Transfer Rates").

3.5 Repeat Acquisition of the Same Receive Data Frame

Generally speaking, the acquisition and DMA transfer of the data for a specific frame of a specific receive buffer should occur once and only once in the entire event sequence of your script. While it may be convenient to re-use the same set of Receive structures and memory allocation at more than one point, this is not a good idea since it can lead to ambiguity as to which set of acquisitions on the hardware system produced the data that currently resides in the receive data buffer being accessed by the system software (for most scripts, there is not a rigid, deterministic timing relationship between the software and hardware event sequences). In addition to the data content ambiguity in an asynchronous script, there can also be DMA handshaking ambiguity if you try to synchronize the script (If I am supposed to wait for DMA transfer complete to a specific frame and buffer, how do I know which DMA to wait on if there are two DMA's to that frame in the script?).

To avoid these pitfalls, Verasonics advises you to avoid duplicate acquisitions to the same frame if at all possible. It is usually quite simple to define a separate frame (or a separate receive buffer) in the script instead of re-using the same one. There are some cases, however, where it may be important to violate this guideline. One obvious example would be the definition of a very large RF data frame to be acquired from a large number of acquisition events, and there is not enough memory space in the hardware system to allow the definition of two separate receive data frames plus two sets of receive descriptors.

If you do choose to repeat acquisition of the same frame and buffer at more than one point in your script, you must re-use the same set of Receive structures for each of the repeat acquisitions of the frame and the number of acquisitions included in the frame must also be identical. If you violate these guidelines the system may interpret one of the acquisitions as a subframe, which could prevent the processing functions from operating on that acquisition. The system software identifies a subframe DMA based on the following criteria:

- The next transferToHost command in the Event sequence must be to the same buffer and frame numbers as the current transferToHost.
- The Receive structures included in the next transferToHost must be different than the Receive structures included in the current transferToHost.

If both of these conditions are met, the system will interpret the current transferToHost as a subframe transfer, and thus the completion of this DMA will not be regarded as the completion of acquisition and DMA of the entire frame. Therefore, if your intent is to repeat acquisition and DMA of the same entire frame more than once in the event sequence, the Receive structures included in each DMA must be identical. This also implies that you cannot use subframe DMA's if you are going to be doing acquisition and DMA of the same frame more than once. (As noted at the beginning of this subsection, the best way to avoid these complexities is to never do acquisition and transfer of the same receive buffer frame at more than one point in the event sequence!)

4. DMA Transfer Rates

The Vantage hardware system can acquire and store per-channel RF data at programmable sample rates over a very wide range, up to a maximum sample rate of 62.5 MHz. The system hardware design has no difficulty acquiring and storing data in the channel group memory buffer at that rate, but the maximum rate at which the system can transfer data from the hardware system to the host computer is more limited. This section explains the system design constraints that limit the maximum DMA transfer data rate and what can be done to optimize the performance of a script within those constraints.

All receive data is transferred from the hardware system to the host computer through DMA transfers over the PCI-express links that connect them. The DMA transfers are managed by DMA control logic built into the system hardware, so they have negligible impact on the processing throughput of the host computer. Similarly, the DMA controllers in the hardware use dedicated logic that runs in parallel with the receive data acquisition functions so the only impact on the hardware acquisition sequence is a single command executed at the point in the sequence where a DMA is to be launched. Once that command has been issued the hardware system is free to continue with acquisition events for the next frame (or subframe). The only other interaction with the DMA controller is at the point where the hardware sequencer is to launch the next DMA; if at that point the previous DMA is still not complete, the hardware event sequence will pause and wait for the previous DMA to complete and then launch the new DMA and resume. The system can only execute one DMA command at a time, and there is no mechanism for queuing up pending DMA commands.

The dominant limit on DMA transfer rates is the achievable data rate on the PCIe links that connect the hardware system to the host computer. There are three sets of links in that chain; whichever of these three is the most restrictive will set the DMA data transfer rate:

1. Within the Vantage hardware system, each channel group of 32 channels has a separate PCIe link to a multiport PCIe switch on the backplane module. The link

from each channel group is 4 lanes at PCIe Gen 2 speed, providing a maximum achievable data transfer rate of approximately 1.2 GByte per second.

2. There is a link with 8 lanes at PCIe Gen 3 speed in the cable that interconnects the hardware system to the host computer. This link goes from the PCIe switch mentioned above to another PCIe bridge device located on the Verasonics host adapter card installed in the host computer. This link has a maximum achievable data transfer rate of approximately 6.6 GByte/sec.
3. The host computer motherboard provides a PCIe link to the expansion slot where the Verasonics host adapter card is installed. If this link can provide either 8 lanes at gen 3 speed or 16 lanes at gen 2 speed, it will be able to handle the full data rate of the link in item 2 above and thus will never restrict the DMA transfer throughput. Note however that the link width and speed available at individual expansion slots varies greatly with the make and model of computer, and typically also varies from slot to slot within the same computer. If the computer is configured incorrectly or cannot support the required 6.6 GByte/sec rate, the system will automatically adapt to the lower rate that is available. This will trigger no errors or warning messages; the only symptom will be slower DMA rates and thus longer pauses if the hardware acquisition has to stop and wait for a DMA to complete. Properly configured host computers (either as provided through Verasonics or purchased per Verasonics guidelines) will be able to support the full 6.6 GByte/sec data rate, and in this case the throughput bottleneck will always be either item 1 or 2 as described above.

The net result is that when fewer than six channel groups are active, the individual channel group links will set the DMA transfer rate (number of active channel groups times 1.2 GByte/sec.). If six or more channel groups are active the shared cable link will limit the overall rate to 6.6 GByte/sec. For the common operating states of 128 or 256 channels active, the typical DMA rates are as follows:

- 128 Channels: 4.9 GByte/sec total or per-channel sample transfer rate of 19 MSamples/sec
- 256 Channels: 6.6 GByte/sec total or per-channel sample transfer rate of 13 MSamples/sec

The total time required to execute a DMA transfer will be the sum of two components: the actual transfer of the data (at the rate cited above) plus a fixed ‘overhead time’ of a few milliseconds representing the time required to initiate the transfer and then signal to the host computer when it has been completed. For a DMA containing a small block of data the overhead time will be a significant fraction of the total, and thus the net DMA transfer rate will be far less than the values cited above. For DMA’s of approximately 64 MBytes or larger the overhead time becomes insignificant and the net transfer rate will be as stated above. In an application where the DMA transfer rate is an important performance factor you should structure the script so the individual DMA’s include 64 MBytes or more, to achieve the maximum throughput. If individual image frames are much smaller than that, it may be advantageous to bundle multiple frames together into a larger 'superframe' for DMA transfer.

The overhead time appears to be proportional to the number of channel groups that are active, at (very roughly) 0.5 milliseconds per channel group. To get a very crude estimate of the total time for a small DMA, calculate the transfer time using the rates described above and then add on the estimated overhead time. Note that for DMA's smaller than a few MBytes the data transfer time will be insignificant compared to the overhead time.

Another factor that must be considered in applications where real-time acquisition rates are critical is jitter in the DMA rate due to variations in DMA overhead time. The overhead time includes interrupt-based interactions with the host computer, and the latency required to service an interrupt can vary dramatically depending on what other higher-priority tasks or interrupts the operating system may be working on. The transfer rates given above are averages based on the execution of a large number of DMA transactions; most of the individual transactions will be a small amount faster than the average and occasionally there will be one that is far far slower due to unpredictable OS activity. It is common for occasional DMA's to take 10 or 20 milliseconds longer than the average interval over a large number of DMA's of the same size. If it is critical that the acquisition sequence never pause to wait for completion of that occasional worst-case DMA, you may have to plan for a very large safety margin above the typical DMA rate. Alternatively, you might be able to structure the event sequence such that no new DMA's are launched during the time-critical acquisition part of the sequence.

Verasonics, Inc.
12016 115th Ave NE
Kirkland, WA 98034
USA