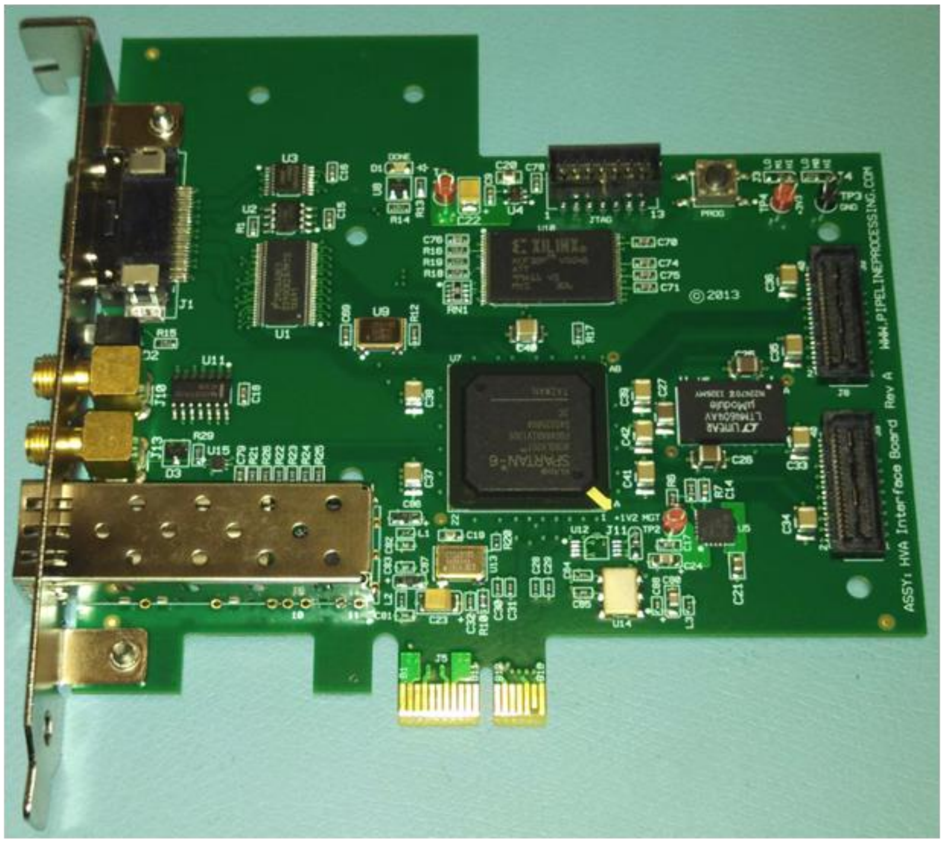
Boston Micro-Machines PCI Express® HVA Interface Board User's Manual Revision 5.XX

FPGA Design 0x2000.05.XX



<u>Terms of Use:</u>

IMPORTANT INFORMATION ON TERMS OF USE AND LEGAL RESTRICTIONS

NOTICE: PLEASE READ THE FOLLOWING INFORMATION CAREFULLY BEFORE USING. BY USING THIS HARDWARE, AND ASSOICATED SOFTWARE, YOU ACCEPT THESE TERMS AND CONDITIONS. IF YOU DO NOT ACCEPT THESE TERMS, DO NOT USE THIS PRODUCT.

1. LIMITED WARRANTY. The PCI Express® HVA Interface Board is warranted against defects in materials and workmanship for (6) months from the date of shipment under normal use and service. Any replacement product will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Customer shall pay expenses for shipment of repaired or replacement Products to and from the vendor.
2. NO OTHER WARRANTIES. Except as expressly set forth above, the products, and associated software, are provided “AS IS” without warranty of any kind, and no other warranties, either expressed or implied are made with respect to the products, including but not limited to any implied warranties of merchantability, fitness for a particular purpose, title or non-infringement or any other warranties that may arise from usage of trade or course of dealing. There is no warranty or guarantee of the use of the products in terms of correctness, accuracy, reliability, or otherwise. There is no warranty or guarantee that the use of the product will be uninterrupted or error free.
3. NO LIABILITY FOR CONSEQUENTIAL DAMAGES. The entire liability of Boston Micro-Machines Corporation (BMC) and its licensors, distributors, and suppliers (including its and their directors, officers, employees, and agents) is set forth above. To the maximum extent permitted by applicable law, in no event shall BMC and its licensors, distributors, and suppliers (including its and their directors, officers, employees, and agents) be liable for any damages, including, but not limited to, any special, direct, indirect, incidental, exemplary, or consequential damages, expenses, lost profits, lost savings, business interruption, lost business information, or any other damages arising out of the use or inability to use the products, even if BMC or its licensors, distributors, and suppliers has been advised of the possibility of such damages. Customer acknowledges that the applicable purchase price or license fee for the products reflects this allocation of risk.
4. WARNING. BMC products are not designed with components and testing for a level or reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. In any application, including the above, reliability of operation of the hardware and software products can be impaired by adverse factors, including but not limited to fluctuations in electrical power supply, computer hardware malfunctions, computer operating system software used to develop an application, installations errors, software and hardware compatibility problems, malfunctions or failures in electronic monitoring or control devices, transient failures of electronic systems (hardware and/or software), unanticipated uses or misuses, or error on the part of the user or applications designer. Customer understands and agrees that BMC has not tested or certified its hardware and software for use in high risk applications including medical life support, nuclear power, mass and air transportation control, or any other potentially life critical uses and makes no assurances that the services are suitable for any high risk activities.

CAUTION: The HVAIF board is a static sensitive electronic device and is susceptible to electrostatic, electromagnetic, magnetic or radioactive fields. Observe proper precautions while handling and inserting into the host computer.

ATTENTION
OBSERVE PRECAUTIONS FOR HANDLING
ELECTROSTATIC SENSITIVE DEVICES

PCI Express® is a registered trademark of PCI-SIG.
Camera Link® is a registered trademark of Automated Imaging Association (AIA)

Table of Contents:

Hardware Revisions:

0x2000.01.00	6/2011	Initial Engineering Release
0x2000.01.01	7/2011	a. Dither Waveform changed from +/-1.0 to 0.0 to 1.0. b. DM Orientation on HVA LUT for HVA32 is on 2^8 address boundaries instead of 2^6.
0x2000.01.02	8/2011	a. Changed Default HVA32 mapping. b. Dither Overflow now sets the result to 0xFFFF and not 0x7FFF.
0x2000.02.00	8/2012	a. Added HVA Bursting Access b. Requires Driver Revision 1.15 and DLL Revision 1.6
0x2000.02.01	8/2013	a. CameraLink Clock Output is ~83 MHz.
0x2000.03.01	9/2013	a. Added Sequence.
0x2000.04.00	10/2013	a. Support for S-driver and Kilo-driver over Fiber.
0x2000.04.01	10/2013	a. Added TTL Output Options.
0x2000.05.01	3/2014	a. Added Ability to support any size DM up to 4K. b. Frame Mode and supporting hardware is removed. Must use Bursting.

I) Introduction:

The PCI Express® (PCIe) HVA Interface Card is designed to interface to most all of the Boston Micro-Machines HVAs. It is a 1-lane, 1st generation duplex link operating at 2.5 Gbps. It is a full height card capable of inserting into any standard size PCIe slot on a computer motherboard, including x1, x4, x8, and x16 slots. The motherboard automatically determines the link width and adjusts accordingly.

The PCI Express® HVA Interface Card is supported under (Windows 7 – 64-bit and Linux 64-bit) Software drivers and low level card access is provided using a pre-compiled DLL in Windows. The PCIe HVA Interface Card only supports 32-bit register reads and writes in PCI Memory space. Byte and or 16-bit accesses are not supported. All addresses must be on a 32-bit boundary. No DMA engine exists.

The PCIe HVA Interface Card has an LED that will blink for 250 ms when a command has been transmitted to the HVA. This is nice visual feedback. A solid green light means that multiple commands are continuing to transmit to the HVA.

Two SMA connectors are on the front panel. One is a 50 Ohm LVTTL output that by default pulses high for 10 micro-second every time a command is sent to the HVA. This can be hooked to a scope for detailed external timing of the system. The other SMA is a LVTTL input that can be used by the sequencing feature on the HVA Interface card.

HVA Commands are transmitted to the HVA out either the CameraLink Port and/or the single duplex Fiber port, depending on which HVA is enabled.

II) Theory of Operation:

The PCIe HVA Interface Card communicates with the HVA using the Camera Link® Base standard and/or a Small Form Factor Pluggable (SFP) Fiber connection. A Mini (HDR) Camera Link® (camera) port is on the board and transmits data to the HVA at a pixel clock rate of 83.33 MHz (12 ns). The fiber link is a 2.5 Gbps full duplex link that uses the Xilinx Aurora protocol. The PCIe card and HVA have been designed such that actuators can be accessed in any random order. There is no forced command order and no forced frame size requirements. Even single actuator "Poke" commands are supported.

There are four ways this PCIe card will write data to the HVA.

1) Single Actuator Poke
2) On board Dither Hardware
3) On board Sequencing Hardware
4) High Speed Bursting

Single Actuator Poke: Software can move a single actuator without moving any of the others or having to re-write all the other actuators.

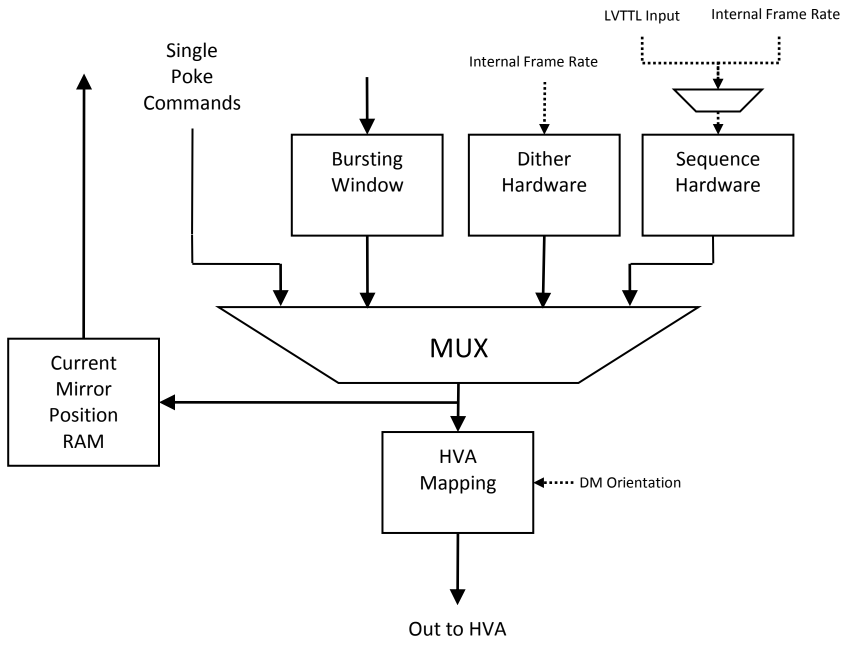
On Board Dither: The PCIe board has the capability of sending a fixed 2-D pattern to the HVA and changing the magnitude of that fixed pattern in a software loadable waveform memory. The dither hardware can synthesize any discrete waveform with elements up to 2048 in length. Software also controls the frame rate of the dither commands. Since the dither hardware does not use the PCIe bus, extremely fast frame rates can be achieved that are limited only by the Camera Link® or Fiber interface. (Note that the HVA may not be able to process commands at rates that can be generated by the dither hardware.)

On Board Sequencing: Software downloads a sequence of several DM patterns and the commands will transmit on the rising edge of the TTL input. Each TTL pulse will send a complete set of commands. When the end of the sequence is reached the first pattern is repeated. The DM commands can also be delayed a software settable time after the TTL pulse is repeated.

Bursting: With this method each actuator has a specific PCIe address and all commands can be transmitted by software very quickly using Write-Combining. Below lists the typical transfer times for various HVA/DM sizes using Window 7 and an inexpensive PC. Transfer times are not guaranteed. Windows 7 is not a real-time operating system.

HVA/DM	**Typical Transfer Time**
HVA32	445 ns
HVA140	1.78 µs
HVA492	10.5 µs – 11.4 µs
HVA952	15 µs – 16.4 µs
HVA1024	17 µs – 19.3 µs
HVA2040	35 µs – 37.3 µs
HVA4096	77 µs – 80 µs

Figure 1: PCIe HVA Interface Board Block Diagram



The actuator addressing on all HVAs have two different forms. Actuators can be thought of as a 2-D array starting with address 1 in the upper left corner and ending with the size of the DM in the lower right corner. In some cases the DM is thought of as a single 1-D frame of N commands. Included in the SDK are several HVA/DM combinations. (Shown below) Users can make their own HVA/DM combinations. See appendix III.

	Orientation	**Output Port Used**
HVA32	6x6	Camera Link
HVA140	12x12	Camera Link
HVA1024	32x32	Fiber
KILO	32x32	Fiber

See Figures 2-4 below for User Space HVA channel addressing for a couple different HVAs. All commands will be mapped to the proper HVA channel using a look-up table. If the deformable mirror is not in the same orientation as the sensor or the algorithm results, software can download a rotated mapping to handle all possible orientations. Any channels that do not exist in the array are also addressable but will not transmit to the HVA. The four corners are actuators that do not exist. Software can write those corners as any other actuator and the PCIe card will strip those commands out and not send them to the HVA.

Figure 2: HVA 32 Channel Addressing (User Space)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Figure 3: HVA 140 Channel Addressing (User Space)

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132
133	134	135	136	137	138	139	140	141	142	143	144

III) Known Issues

1) Some x16 slots on older computers are dedicated graphics slots do not operate any PCI Express® card other than a video graphics cards. If this occurs try another slot or check to be sure the computer has the latest BIOS.
2) Bursting is accomplished by using Write-Combining. This allows for software to do a mem-copy to physical PCIe address space. While this is a quick way to send data to the PCIe card without a DMA engine, it does not guarantee that the order of the commands. The PCIe card operates without errors when this happens because the DM supports random access.

IV) Matlab Software Development Kit:

Included with the Boston Micro-Machine's PCI Camera Link® Output board is a full suite of high level Matlab functions. All the functions, and the full source code, can be found in the Boston Micro-Machines directory. All the functions can be separated into three separate groups; Setup, Status, and Run-Time. The Setup functions are used to configure the HVAIF board and prepare the hardware and HVA/DM for operation. Status functions can be used to read and report the current state of the HVA and DM. The limited Run-Time functions take the algorithm results and transmit them to the HVA. Run-Time functions are lean and written to send the commands as quick as possible with little overhead. Each function has a file with the same name and the extension .m. All functions can be called from a Matlab script. All parameters are the native Matlab double precision floating point or Strings where appropriate. The driver enumerates the PCIe boards starting with 1. In all functions **BrdNum** must be a number between 1 and N. Board 0 does not exist and will return an error. Table A below shows all the Matlab functions included in the SDK.

Function Name	**Group**
AbortFramedData	Setup
ClearHVA	Setup
DisableDither	Setup
DisableSequence	Setup
EnableDither	Setup
EnableSequence	Setup
PokeDM	Setup
SetDMorientation	Setup
SetHVA	Setup
SetUpHVA	Setup
ShutdownHVA	Setup
WriteDitherGains	Setup
WriteDitherWave	Setup
WriteHVALUT	Setup
WriteSequence	Setup
Decode_Error	Status
GetCurrentHVAmode	Status
GetNumDevices	Status
ReadDMorientation	Status
ReadFiberLinkState	Status
ReadHVAccStatus	Status
ReadHVALUT	Status
ReadHVAstate	Status
ReadHWRevision	Status
ReadKiloStatus	Status
ReadSequenceRate	Status
RetrieveHVAInfo	Status
BurstHVA32Frame1D	Run-Time
BurstHVA32Frame2D	Run-Time
BurstHVA137Frame1D	Run-Time
BurstHVA137Frame2D	Run-Time

BurstHVA140Frame1D	Run-Time
BurstHVA140Frame2D	Run-Time
BurstHVA492Frame1D	Run-Time
BurstHVA492Frame2D	Run-Time
BurstHVA952Frame1D	Run-Time
BurstHVA1024Frame1D	Run-Time
BurstHVA1024Frame2D	Run-Time
BurstHVA2040Frame1D	Run-Time
BurstHVA4096Frame1D	Run-Time

In addition to the functions listed above there are a few test functions. The TTL output can be used for several purposes. The functions below also allow software to set the TTL output pulse to a minimum pulse width if desired.

Function Name	**Description**
TTLo_AnyCommand	Pulse when any command is sent to the Mirror.
TTLo_FVAL	TTLo is high when FVAL is high.
TTLo_SpecificActuator	Pulse when a command is sent to a designated actuator.
TTLo_SpecificCommand	Pulse when a designated command is sent to any actuator.
TTLo_SpecificActuatorCommand	Pulse when a designated command is sent to a designated actuator.

Examples of TTL output use.

1) TTLo_Specific Actuator: The TTL output could pulse when the last command to the mirror is written.
2) TTLo_SpecificCommand: The TTL output could pulse when a DM command saturates at the maximum value, signifying a possible unstable control system.

[error] = AbortFramedData(BrdNum)

This function is used to recover from an error condition. Call this function to end the current frame that is transmitting out of the HVAIF board. This error occurs when the incorrect amount of data is written to the card using the BurstHVAXXFrameYD functions. The amount of data must equal the size of the DM.

[error] = ClearHVA(BrdNum)

Resets all the actuators on the DM to zero.

[error] = DisableDither(BrdNum)

Use this function to turn off the Hardware Dither feature on the HVAIF Board. See Appendix I for more information on dithering the mirror.

[error] = DisableSequence(BrdNum)

Use this function to turn off the Hardware Sequencing feature on the HVAIF Board. See Appendix II for more information on sequencing.

[error] = EnableDither(BrdNum, FRRate)

Use this function to enable the Hardware Dither Feature on the HVAIF Board. This function accepts the desired Frame Rate. FRRate can range from 0 Hz to rates faster than the HVA can operate. This function should not be used until the Dither Waveform and Pattern have been configured. See Appendix I for more information on dithering the mirror.

[error] = EnableSequence(BrdNum, FRRate)

Use this function to enable the Sequencing hardware on the HVAIF Board. This function accepts the desired Frame Rate. FRRate can range from 0 Hz to rates faster than the HVA can operate. This function should not be used until the WriteSequence function has been called. See Appendix II for more information on sequencing.

[error] = PokeDM(BrdNum, ActNum, Data)

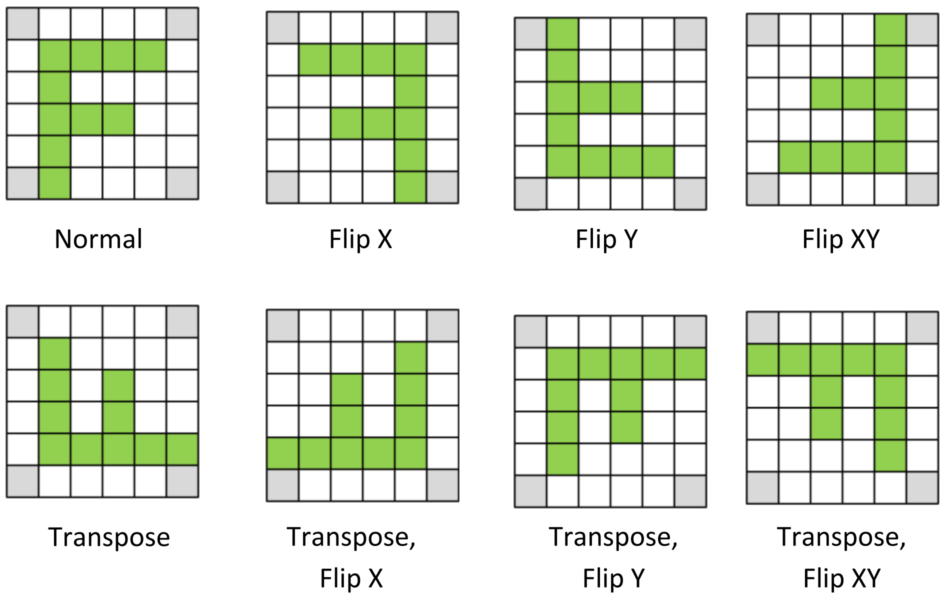
Use this function to poke any single actuator on the DM. ActNum can range from 1 to N, (depending on the current mode) and Data can range from 0 to 65535. See figures 2 or 3 for the Actuator position of ActNum for the HVA32 and HVA140.

[error] = SetDMorientation(BrdNum, OrienSelect)

Use this function to change the DM to WFS orientation. All eight possible 2D orientations are available. OrienSelect is a case sensitive string. See the table and Figure 4 below for the definition of each choice.

Normal
FlipX
FlipY
FlipXY
Transpose
Transpose, FlipX
Transpose, FlipY
Transpose, FlipXY

Figure 4: All possible DM orientations. (6x6 case)



[error] = SetHVA(BrdNum, Value)

Sofware can use SetHVA to poke the entire mirror to a known position. The current state of the mirror is first checked and then a for-loop is used to poke each actuator individually. This function is slow and should not be used in real-time processing.

[error] = SetUpHVA(BrdNum, HVAType)

SetUpHVA must be called at the beginning of any Matlab program. This does three things.

1) Loads the ppcdll.
2) Sets the HVA into the desired mode of operation.
3) Reads the Hardware Revision to check that this board is a Boston Micro-Machines HVAIF board.

HVAType is a case sensitive string that represents the HVA/DM. The constants loaded from BMCHVA_config_v50.mat contain all the information required for all currently support HVA/DMs.

ShutdownHVA should be called when the Matlab program finishes to release the DLL.

[error] = ShutdownHVA()

ShutdownHVA is called at the end of the Matlab execution. It will release the ppcdll.

[error] = WriteDitherGains(BrdNum, Array1D)

This function is used in the hardware dithering configuration. The dither magnitude in HVA counts is downloaded to internal RAM. Each actuator has its own dither gain. Array1D must be a one dimensional array of size equal to the current HVA mode. Array1D can range from 0 to 65535. See Appendix I for more information on Dithering.

[error] = WriteDitherWave(BrdNum, Waveform)

This function is also used in the hardware dithering configuration. The dither waveform is a one dimensional array of numbers between 0.0 and 1.0 that create a discrete waveform. Hardware repeats this waveform indefinitely when enabled. The waveform can be between 2 and 2048 in length. See Appendix I for more information on Dithering.

[error] = WriteHVALUT (BrdNum, Array, Mask)

This is an advanced function reassigns the User Space actuator addressing (Figures 2 and 3) to the actual HVA DAC addresses. This function should never be required unless the delivered HVA is a special custom variant with alternate mapping. If this register is accidently written, ResetHVALUT will return the mapping to the power-up state. The Array input must be 2D and the correct size for the HVA mode. Mask is also 2D and the same size as Array. When Mask(m,n) is zero, then that channel does not exist on the DM and the appropriate bit will be set in hardware to prevent writing to that channel.

[error] = WriteSequence (BrdNum, sequenceNxNxM, delay)

This single function configures the sequencing hardware. Software generates a NxNxM array, where the DM size is NxN and the length of the sequence is M. The delay parameter will delay the output up to 0.25 seconds. The entire length of the sequence (NxNxM) cannot be greater than 4095.

[error_string] = Decode_Error (code)

If software wants more information about an error code returned from any Matlab function, this routine will provide a string description of the error. See Table B for information about the types of errors possible.

Table B: Possible Error Codes

Code	**Function**	**Description**
0	All	No Error
12	All	Board not Found – Range = 1 to N
1000	SetUpHVA	Board is not a BMC HVAIF Board
1001	SetUpHVA	Unknown HVA Type
1002	SetDMorientation	Invalid Orientation Selection
1003	ReadDMorientation	Invalid DM Orientation Response
1006	WriteHVALUT	Invalid HVA LUT

1007	PokeDM	ActNum invalid for HVA Type (1 to NxN)
1008	WriteDitherGains	Invalid Dither Gain Array Size (1D)
1009	WriteDitherGains	Invalid Dither Gain Array Value (0 to 65535)
1010	WriteDitherWave	Waveform Array too large (2048 maximum)
1011	WriteDitherWave	Invalid Waveform Array Value (0.0 to 1.0)
1012	EnableDither	Dither Rate must be positive.
1013	WriteSequence	Invalid Actuator command (0 to 65535)
1014	WriteSequence	Various Errors
1015	WriteSequence	Delay out of range (0 – 0.25 seconds)
1016	EnableSequence	Invalid Sequence Rate (Must be Positive)
1017	TTLo_*	TTL Pulse width out of range (0-700 µs)
1018	TTLo_*	TTL Pulse DAC channel out of range (0-1023)
1019	TTLo_*	TTL Pulse DAC command out of range (0-65535)
1020	Various	Not in correct HVA Mode.

[error Mode] = GetCurrentHVAmode (BrdNum)

Call this function to see the current mode of the HVAIF board. A structure is returned showing the current state of the HVAIF board. The idnum can be used to determine which HVA/DM is currently active.

[error NumDev] = GetNumDevices ()

Call this function to see how many PCIe boards are currently in the host PC. If the NumDev response is zero then either the card is not present in the PC or the driver has yet to be installed on the machine.

[error Orien] = ReadDMorientation (BrdNum)

Call this function to see the current orientation of the DM. Orien Responses can be one of the following.

Normal
FlipX
FlipY
FlipXY
Transpose
Transpose, FlipX
Transpose, FlipY
Transpose, FlipXY

[error Fiberstatus] = ReadFiberLinkState(BrdNum)

Reads the current link state and returns a structure.

Table C: Fiber State

	Description
Sfp_preset	Fiber Module Present? 1 = Present.
Sfp_txfault	SFP Transmitter Fault
Sfp_los	SFP loss of light.
Channel_up	Aurora Channel is operating properly
Lane_up	Aurora Lane is operating properly
Soft_error	Aurora Link Error
Hard_error	Aurora Link Error

[error HVAstatus] = ReadHVAccStatus(BrdNum)

The HVA uses the Camera Link® CC(4:1) signals as generic status bits. Software can use this function to read those status indicators. HVAstatus returns a structure. See Table D.

Table D: HVAstatus bit assignment.

CC#	Num	Bit	Description
CC1	1	HVAstatus(0)	High Voltage On
CC2	2	HVAstatus(1)	Data Dropped*
CC3	4	HVAstatus(2)	DACs Busy*
CC4	8	HVAstatus(3)	Any Error Since Power-On

* These bits are active for only a short period of time. Software reading these bits as a ‘1’ is highly unlikely.

[error LUT] = ReadHVALUT(BrdNum)

Call this function to read the current state of the HVA LUT. Only the first 144 or 36 values are read representing the Normal Orientation. This function is primarily used for factory testing.

[error State] = ReadHVAstate(BrdNum)

Software can call this function at any time to get the current position of the mirror. The current mode of the HVAIF board is checked and a 1-Dimension array of the proper length is returned in State. Higher level software must resize the 1-D array into the proper 2D dimensions.

[error HWRev] = ReadHWRevision(BrdNum)

Software can call this function to read the Hardware revision of the HVAIF board. This is a constant set at the factory. The value is converted to a hexadecimal string. The Hardware Revision consists of a Design ID, Major Revision, Minor Revision.

Design ID	Major	Minor
2000	05	01

2000 is the Revision Number for a BMC HVAIF Board.

[error Status] = ReadKiloStatus(BrdNum)

Returns a structure with the following status:

- DAC Overflow
- Address Out of Range
- FIFO Half Full

[error FRate] = ReadSequenceRate(BrdNum)

Software can call this to read the current incoming TTL frame period. If the internal frame rate generator is selected then this register will report the current SW selectable frame rate.

[error] = BurstHVA36Frame1D(BrdNum, DMData)

Use this function to command a 32 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x36 or 36x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA36Frame2D(BrdNum, DMData)

Use this function to command a 32 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 6x6. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

[error] = BurstHVA137Frame1D(BrdNum, DMData)

Use this function to command a 137 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x169 or 169x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA137Frame2D(BrdNum, DMData)

Use this function to command a 137 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 13x13. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA140Frame1D(BrdNum, DMData)

Use this function to command a 140 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x144 or 144x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA140Frame2D(BrdNum, DMData)

Use this function to command a 140 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 12x12. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

[error] = BurstHVA492Frame1D(BrdNum, DMData)

Use this function to command a 492 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x576 or 576x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA492Frame2D(BrdNum, DMData)

Use this function to command a 492 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 24x24. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

[error] = BurstHVA952Frame1D(BrdNum, DMData)

Use this function to command a 952 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x952 or 952x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

[error] = BurstHVA1024Frame1D(BrdNum, DMData)

Use this function to command a 1024 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x1024 or 1024x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will “roll-over”.

[error] = BurstHVA1024Frame2D(BrdNum, DMData)

Use this function to command a 1024 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 32x32. DMData also must be between 0 and 65535. Numbers greater than 65535 will “roll-over”. Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

[error] = BurstHVA2040Frame1D(BrdNum, DMData)

Use this function to command a 2040 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x2040 or 2040x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will “roll-over”.

[error] = BurstHVA4096Frame1D(BrdNum, DMData)

Use this function to command a 4096 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. DMData has to be exactly 1x4096 or 4096x1. DMData also must be between 0 and 65535. Numbers greater than 65535 will “roll-over”.

V) LabView Software Development Kit:

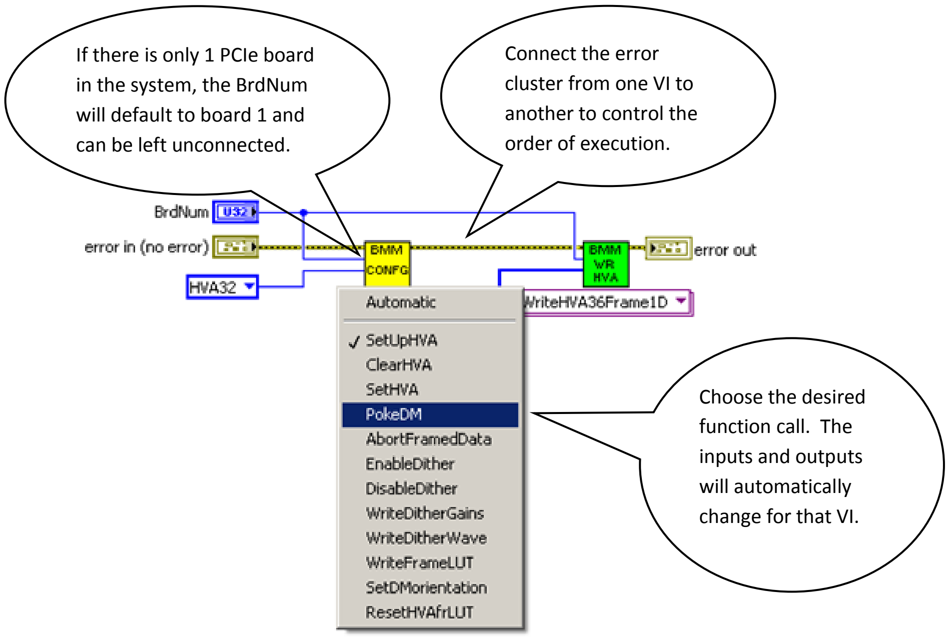
Included with the Boston Micro-Machine's PCI Camera Link® Output board is a full suite of high level LabView functions. All the functions, and the full source code, can be found in the Boston Micro-Machines directory. All the functions are separated into three separate groups; Configuration, Status, and Run-Time. The Configuration functions are used to configure the HVAIF board and prepare the hardware and HVA/DM for operation. Status functions can be used to read and report the current state of the HVA and DM. The limited Run-Time functions take the algorithm results and transmit them to the HVA. Run-Time functions are lean and written to send the commands as quick as possible with little overhead or error checking. Each function has a file with the same name and the extension .vi. All functions are grouped into one of three polymorphic VIs. The VIs are written using 64-bit LabView 2012 v12. Table D lists all the available VIs and their corresponding polymorphic VI.

Function Name	**Polymorphic VI**
AbortFramedData	BMC_config.vi
ClearHVA	BMC_config.vi
DisableDither	BMC_config.vi
DisableSequence	BMC_config.vi
EnableDither	BMC_config.vi
EnableSequence	BMC_config.vi
PokeDM	BMC_config.vi
SetDMorientation	BMC_config.vi
SetHVA	BMC_config.vi
SetUpHVA	BMC_config.vi
WriteDitherGains	BMC_config.vi
WriteDitherWave	BMC_config.vi
WriteHVALUT	BMC_config.vi
WriteSequence	BMC_config.vi
TTLo_AnyCommand	BMC_config.vi
TTLo_FVAL	BMC_config.vi
TTLo_SpecificActuator	BMC_config.vi
TTLo_SpecificActuatorCommand	BMC_config.vi
TTLo_SpecificCommand	BMC_config.vi
GetCurrentHVAmode	BMC_status.vi
GetDrvRev	BMC_status.vi
GetNumDevices	BMC_status.vi
GetSWRev	BMC_status.vi
ReadDMorientation	BMC_status.vi
ReadFiberLinkState	BMC_status.vi
ReadHVAccStatus	BMC_status.vi
ReadKiloStatus	BMC_status.vi
ReadHVALUT	BMC_status.vi
ReadHVAstate	BMC_status.vi
ReadHWRevision	BMC_status.vi
ReadSequenceRate	BMC_status.vi
RetrieveHVAInfo	BMC_status.vi
BurstHVA36Frame1D	BMC_runtime.vi

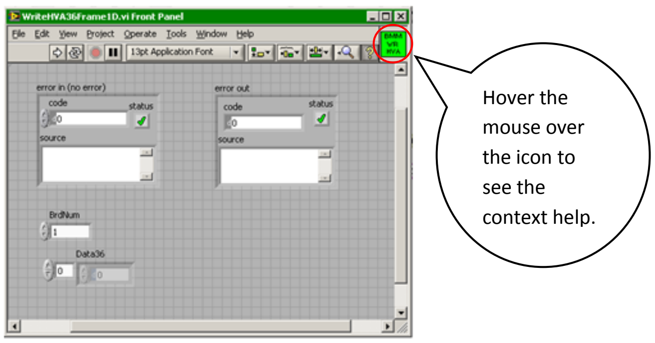
BurstHVA36Frame2D	BMC_runtime.vi
BurstHVA137Frame1D	BMC_runtime.vi
BurstHVA137Frame2D	BMC_runtime.vi
BurstHVA144Frame1D	BMC_runtime.vi
BurstHVA144Frame2D	BMC_runtime.vi
BurstHVA492Frame1D	BMC_runtime.vi
BurstHVA492Frame2D	BMC_runtime.vi
BurstHVA952Frame2D	BMC_runtime.vi
BurstHVA1024Frame1D	BMC_runtime.vi
BurstHVA1024Frame2D	BMC_runtime.vi
BurstHVA2040Frame1D	BMC_runtime.vi
BurstHVA4096Frame1D	BMC_runtime.vi

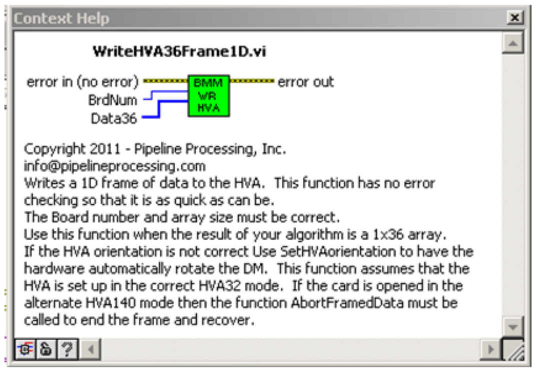
To call one of the functions, just drag and drop one of the three polymorphic VIs into the back-panel and use the drop down to select which function to access. To control the order of execution tie the error output of one block to the error input of the next. Without connecting the error signals, the user has no control of the order of operations and it may be different each time you run the VI.

See the Figure 5 Below for an example of the Configuration polymorphic VI and the possible functions that can be called.


If there is only 1 PCIe board in the system, the BrdNum will default to board 1 and can be left unconnected.
Connect the error cluster from one VI to another to control the order of execution.
Choose the desired function call. The inputs and outputs will automatically change for that VI.
BrdNum
U32
error in (no error)
HVA32
BMM CONFG
BMM WR HVA
error out
WriteHVA36Frame1D
Automatic
SetUpHVA
ClearHVA
SetHVA
PokeDM
AbortFramedData
EnableDither
DisableDither
WriteDitherGains
WriteDitherWave
WriteFrameLUT
SetDMorientation
ResetHVAfrLUT

Context Help: If additional information on the functionality of a VI is required, double click on the polymorphic VI to open the selected sub-VI. Then hover the mouse pointer over the icon and the help for that VI is displayed. See the figure 6 below.


WriteHVA36Frame1D.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt Application Font
error in (no error)
code
status
source
error out
BrdNum
Data36
BMM WR HVA
Hover the mouse over the icon to see the context help.



WriteHVA36Frame1D.vi

Copyright 2011 - Pipeline Processing, Inc.
info@pipelineprocessing.com
Writes a 1D frame of data to the HVA. This function has no error checking so that it is as quick as can be.
The Board number and array size must be correct.
Use this function when the result of your algorithm is a 1x36 array.
If the HVA orientation is not correct Use SetHVAorientation to have the hardware automatically rotate the DM. This function assumes that the HVA is set up in the correct HVA32 mode. If the card is opened in the alternate HVA140 mode then the function AbortFramedData must be called to end the frame and recover.

Error Handling: Besides controlling the execution order, chaining the error clusters together allow for the notification of errors after program execution. When the program execution stops, the last error reported will be in the error out cluster. Both the error code and a string description are available in the cluster. Table E lists the errors, their source, and description.

Table E: Possible Error Codes

Code	Function	Description
0	All	No Error
12	All	Board not Found – Range = 1 to N (BMC DLL)
1000	SetUpHVA	Board is not a BMC HVAIF Board
1001	RetrieveHVAInfo ReadHVAState	Unknown HVA/DM
1003	ReadDMorientation	Invalid DM Orientation Response
1006	WriteHVALUT	Invalid HVA LUT
1007	PokeDM	ActNum invalid for HVA Type (1 to NxN)
1008	WriteDitherGains	Invalid Dither Gain Array Size
1009	WriteDitherGains	Invalid Dither Gain Array Value (0 to 65535)
1010	WriteDitherWave	Waveform Array too large (2048 maximum)
1011	WriteDitherWave	Invalid Waveform Array Value (0.0 to 1.0)
1012	EnableDither	Dither Rate must be positive.
1013	WriteSequence	Invalid Actuator command (0 to 65535)
1014	WriteSequence	Various
1015	WriteSequence	Delay out of range (0 – 0.25 seconds)
1017	Any TTLo Function	Pulse Width Invalid
1019	TTLo_SpecificActuator TTLo_SpecificActuatorCommand	Invalid Actuator Command
1020	SetDMorientation WriteSequence	Not in Correct HVA Mode

AbortFramedData(BrdNum)

This function is used to recover from an error condition. Call this function to end the current frame that is transmitting out of the HVAIF board. This error occurs when the incorrect amount of data is written to the card using the BurstHVAXXFrameYD functions. The amount of data must equal the size of the DM.

ClearHVA(BrdNum)

Resets all the actuators on the DM to zero.

DisableDither(BrdNum)

Use this function to turn off the Hardware Dither feature on the HVAIF Board. See Appendix I for more information on dithering the mirror.

DisableSequence(BrdNum)

Use this function to turn off the Hardware Sequencing feature on the HVAIF Board. See Appendix II for more information on sequencing.

EnableDither(BrdNum, FrameRate)

Use this function to enable the Hardware Dither Feature on the HVAIF Board. This function accepts the desired Frame Rate. FrameRate can range from 0 Hz to rates faster than the HVA can operate. This function should not be used until the Dither Waveform and Pattern have been configured. See Appendix I for more information on dithering the mirror.

EnableSequence(BrdNum, FrameRate)

Use this function to enable the Sequencing hardware on the HVAIF Board. This function accepts the desired Frame Rate. FrameRate can range from 0 Hz to rates faster than the HVA can operate. This function should not be used until the WriteSequence function has been called. See Appendix II for more information on sequencing.

PokeDM(BrdNum, ActNum, Data)

Use this function to poke any single actuator on the DM. ActNum can range from 1 to 36 or 144, (depending on the current mode) and Data can range from 0 to 65535. See figures 2 or 3 for the Actuator position of ActNum.

SetDMorientation(BrdNum, HVAType, DM Orientation)

Use this function to change the DM to WFS orientation. All eight possible 2D orientations are available. DM Orientation is a case sensitive string. See the table below for the possibilities.

Normal
FlipX
FlipY
FlipXY
Transpose
Transpose, FlipX
Transpose, FlipY
Transpose, FlipXY

HVAType is a case sensitive string that matches the current mode of the HVA.

SetHVA(BrdNum, Data)

Sofware can use SetHVA to poke the entire mirror to a known position. The current state of the mirror is first checked and then a for-loop is used to poke each actuator individually. This function is slow and should not be used in real-time processing.

SetUpHVA(BrdNum, HVAType)

SetUpHVA must be called at the beginning of any Labview program. This does two things.

1) Sets the HVA into the desired mode of operation.
2) Reads the Hardware Revision to check that this board is a Boston Micro-Machines HVAIF board.

HVAType is a case sensitive string that represents the HVA/DM. The constants loaded from BMCHVA_config_v50.vi contain all the information required for all currently supported HVA/DMs.

WriteDitherGains(BrdNum, DM Pattern1D)

This function is used in the hardware dithering configuration. The dither magnitude in HVA counts is downloaded to internal RAM. Each actuator has its own dither gain. DM Pattern1D must be a one dimensional array of size equal to the current HVA mode. DM Pattern1D can range from 0 to 65535. See Appendix I for more information on Dithering.

WriteDitherWave(BrdNum, Waveform)

This function is also used in the hardware dithering configuration. The dither waveform is a one dimensional array of numbers between 0.0 and 1.0 that create a discrete waveform. Hardware repeats this waveform indefinitely when enabled. The waveform can be between 2 and 2048 in length. See Appendix I for more information on Dithering.

WriteHVALUT (BrdNum, LutArray2D, ActiveMap2D)

This is an advanced function reassigns the User Space actuator addressing (Figures 2 and 3) to the actual HVA DAC addresses. This function should never be required unless the delivered HVA is a special custom variant with alternate mapping. The LutArray2D input must be 2D and the correct size for the HVA mode. ActiveMap2D is also 2D and the same size as Array. When ActiveMap2D (m,n) is zero, then that channel does not exist on the DM and the appropriate bit will be set in hardware to prevent writing to that channel.

WriteSequence (BrdNum, SequenceMxRxC, Mask2D, Delay)

This single function configures the sequencing hardware. Software generates a MxRxC array, where the DM size is RxC and the length of the sequence is M frames. The Mask2D can be used to only send commands to a subset of HVA channels. The delay parameter will delay the output up to 0.25 seconds. The entire length of the sequence cannot be greater than 4095.

BMCHVA_config_v50

This single function returns an array of clusters for every known HVA Type.

In addition to the functions listed above there are a few test functions. The TTL output can be used for several purposes. The functions below also allow software to set the TTL output pulse to a minimum pulse width if desired.

Function Name	**Description**
TTLo_AnyCommand	Pulse when any command is sent to the Mirror.
TTLo_FVAL	TTLo is high when FVAL is high.
TTLo_SpecificActuator	Pulse when a command is sent to a designated actuator.
TTLo_SpecificCommand	Pulse when a designated command is sent to any actuator.
TTLo_SpecificActuatorCommand	Pulse when a designated command is sent to a designated actuator.

GetCurrentHVAmode (BrdNum)

Call this function to see the current mode of the HVAIF board. A cluster is returned with the following elements.

HVA Type: string
Idnum: Unique ID number (0-31) to represent which HVA/DM is currently in use.
Use Camlink: HVA uses Camera Link® port.
Use Fiber: HVA uses Fiber port.
Fiber Mode: 0 = S-driver, 1 = CBI
Burst Mode: 0 (1 is currently not supported)
Size: Number of actuators in HVA.

GetDrvRev()

This function will query the driver for its software revision.

GetNumDevices ()

Call this function to see how many PCIe boards are currently in the host PC. If the NumDev response is zero then either the card is not present in the PC or the driver has yet to be installed on the machine.

GetSWRev()

This function will query the DLL for its software revision.

ReadDMorientation (BrdNum)

Call this function to see the current orientation of the DM. Responses can be one of the following strings.

Normal
FlipX
FlipY
FlipXY
Transpose
Transpose, FlipX
Transpose, FlipY
Transpose, FlipXY

ReadFiberLinkState(BrdNum)

Reads the current link state and returns a Status Cluster.

Table F: Fiber State

	Description
Sfp_preset	Fiber Module Present? 1 = Present.
Sfp_txfault	SFP Transmitter Fault
Sfp_los	SFP loss of light.
Channel_up	Aurora Channel is operating properly
Lane_up	Aurora Lane is operating properly
Soft_error	Aurora Link Error
Hard_error	Aurora Link Error

ReadHVAccStatus(BrdNum)

The HVA uses the Camera Link® CC(4:1) signals as generic status bits. Software can use this function to read those status indicators. HVAstatus is the sum of the set bits in the column titled NUM in Table F.

Table G: HVAstatus bit assignment.

CC#	Num	Bit	Description
CC1	1	HVAstatus(0)	High Voltage On
CC2	2	HVAstatus(1)	Data Dropped*
CC3	4	HVAstatus(2)	DACs Busy*
CC4	8	HVAstatus(3)	Any Error Since Power-On

* These bits are active for only a short period of time. Software reading these bits as a '1' is highly unlikely.

ReadKiloStatus(BrdNum)

Returns a cluster with the following status:

DAC Overflow
Address Out of Range
FIFO Half Full

ReadHVALUT(BrdNum)

Call this function to read the current state of the HVA LUT. The entire 4K memory array is read regardless of the HVA configuration. If a value is 65536 (0x10000) then that Actuator number is ignored.

ReadHVAstate(BrdNum)

Software can call this function at any time to get the current position of the mirror. The current mode of the HVAIF board is checked and the Dimensions of the returned State will be the proper 2D size.

ReadHWRevision(BrdNum)

Software can call this function to read the Hardware revision of the HVAIF board. This is a constant set at the factory. The value is converted to a hexadecimal string. The Hardware Revision consists of a Design ID, Major Revision, Minor Revision.

Design ID	Major	Minor
2000	05	01

2000 is the Revision Number for a BMC HVAIF Board.

ReadSequenceRate(BrdNum)

Software can call this to read the current incoming TTL frame period. If the internal frame rate generator is selected then this function will report the current SW selectable frame rate.

RetrieveHVAInfo(HVA Type)

Send in a string and this function will give you all the parameters for that HVA/DM. This function uses BMCHVA_config_v50.vi. An error will occur if the HVA is unknown.

BurstHVA36Frame1D(BrdNum, Data36)

Use this function to command a 32 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data36 has to be exactly 1x36 or 36x1. Data36 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA36Frame2D(BrdNum, Data36)

Use this function to command a 32 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. Data36 has to be exactly 6x6. Data36 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

BurstHVA137Frame1D(BrdNum, Data169)

Use this function to command a 137 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data169 has to be exactly 1x169 or 169x1. Data169 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA137Frame2D(BrdNum, Data169)

Use this function to command a 137 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. Data169 has to be exactly 13x13. Data169 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA140Frame1D(BrdNum, Data144)

Use this function to command a 140 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data144 has to be exactly 1x144 or 144x1. Data144 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA140Frame2D(BrdNum, Data144)

Use this function to command a 140 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. Data144 has to be exactly 12x12. Data144 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

BurstHVA492Frame1D(BrdNum, Data576)

Use this function to command a 492 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data576 has to be exactly 1x576 or 576x1. Data576 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA492Frame2D(BrdNum, Data576)

Use this function to command a 492 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. Data576 has to be exactly 24x24. Data576 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

BurstHVA952Frame1D(BrdNum, Data952)

Use this function to command a 952 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data952 has to be exactly 1x952 or 952x1. Data952 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA1024Frame1D(BrdNum, Data1024)

Use this function to command a 1024 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data1024 has to be exactly 1x1024 or 1024x1. Data1024 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

BurstHVA1024Frame2D(BrdNum, Data1024)

Use this function to command a 1024 channel mirror during run time if the algorithm produces results which are two dimensional. This function is meant to be very fast with no error checking. Data1024 has to be exactly 32x32. Data1024 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over". Matlab uses column-major ordering so if this function is used, the DM orientation may have to be changed to one of the transpose options.

BurstHVA2040Frame1D(BrdNum, Data2040)

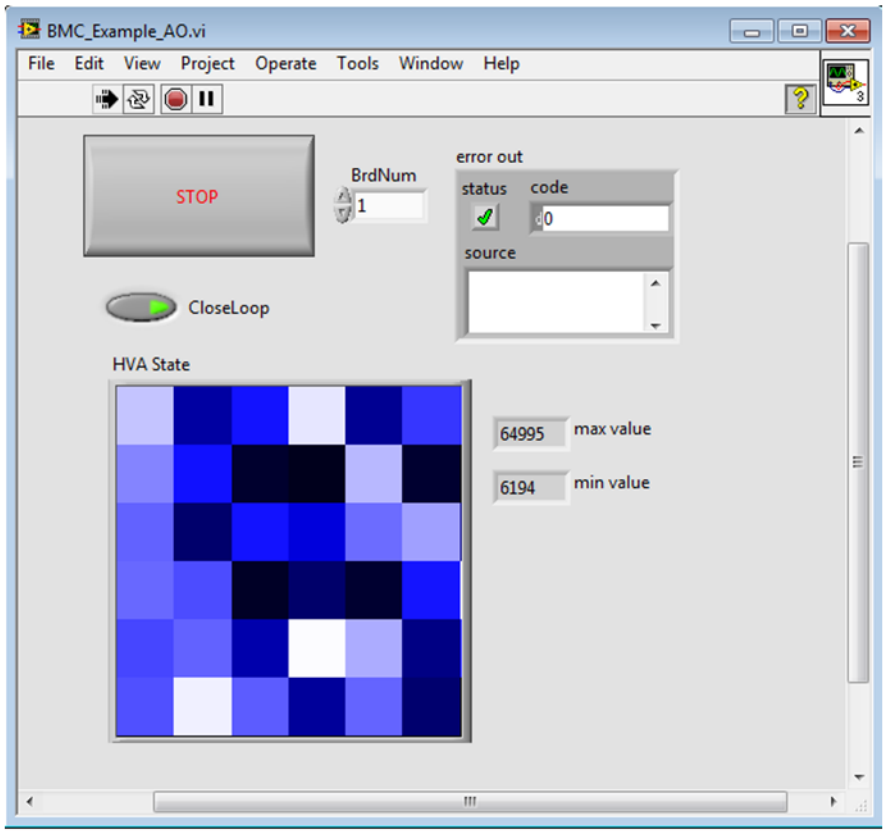
Use this function to command a 2040 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data2040 has to be exactly 1x2040 or 2040x1. Data2040 also must be between 0 and 65535. Numbers greater than 65535 will "roll-over".

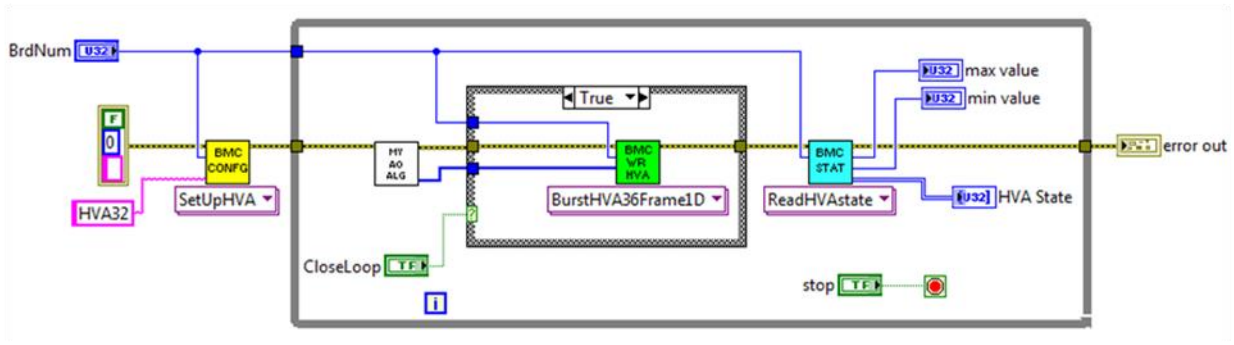
BurstHVA4096Frame1D(BrdNum, Data4096)

Use this function to command a 4096 channel mirror during run time if the algorithm produces results which are one dimensional. This function is meant to be very fast with no error checking. Data4096 has to be exactly 1x4096 or 4096x1. Data4096 also must be between 0 and 65535. Numbers greater than 65535 will “roll-over”.

Example Top Level VI:

The next figure is an example of a simple Lab View based AO system. Every Top Level VI should start with SetUpHVA to ensure that the HVAIF card is set for the proper HVA. Then inside the 'while' loop the sub-VI titled "MY AO ALG" grabs the sensor data and calculates new HVA commands. Then if the control loop is enabled, those commands are sent to the HVA using one of the Run-time polymorphic VIs. If desired, software can read the state of the mirror for display purposes. Both the front and back panel of this example VI are shown. Figure 7.





Low Level DLL Access:

Also included with the LabView software development kit is an additional poly-morphic VI with low level DLL access. These low level functions are utilized in the higher-level VIs but are also available for the user. If an advanced user wants low-level register access to the HVAIF board, use these VIs. In depth knowledge of the hardware and its registers are required when using this set of VIs. Table G lists the low-level VIs. All inputs to the low level VIs are unsigned 32-bit values.

Function Name	Polymorphic VI
BMCClrBits	BMC.vi
BMCClrBitsWait	BMC.vi
BMCLoadMem	BMC.vi
BMCRdReg	BMC.vi
BMCReadMem	BMC.vi
BMCSetBits	BMC.vi
BMCSetBitsWait	BMC.vi
BMCWRReg	BMC.vi

Error = BMCClrBits(BrdNum, RegNum, Data)

Clears the bits which are set as a ‘1’ in the RegNum.

Error = BMCClrBitsWait(BrdNum, RegNum, Data, timeout)

Clears the bits which are set as a ‘1’ in the RegNum and waits the timeout period until those bits are read back as cleared.

Error = BMCLoadMem(BrdNum, RegNum, Increment, Num32, Data)

Writes an array of data to RegNum. The number of values written is Num32. When Increment is TRUE the address is incremented for each write. When Increment is FALSE the same address is written over and over.

Error = BMCRdReg(BrdNum, RegNum, Data)

Reads a register and returns the result in Data.

Error = BMCReadMem(BrdNum, RegNum, Increment, Num32, Data)

Reads an array of data to RegNum. The number of values read is Num32. When Increment is TRUE the address is incremented for each write. When Increment is FALSE the same address is written over and over. The results are returned in the array Data. User must create and provide the Data array for this VI to use.

Error = BMCSetBits(BrdNum, RegNum, Data)

Sets the bits which are set as a ‘1’ in the RegNum.

Error = BMCSetBitsWait(BrdNum, RegNum, Data, timeout)

Sets the bits which are set as a '1' in the RegNum and waits the timeout period until those bits are read back as set.

Error = BMCWrReg(BrdNum, RegNum, Data)

Writes Data to the register RegNum.

User Lab View Library:

It is possible to create a user library so that the developer has access to the BMC VIs from the pop-up function panel. This can make it easier to use the BMC VIs. Follow these steps to create a BMC user library.

1) Open a Windows Explorer Window and navigate to:

   C:\Program Files\National Instruments\LabVIEW 2012\user.lib\

2) Create a new directory named BMCv4.0
3) Copy all the files and directories from:

   C:\Program Files\BostonMicro\LabView\v5.0\

   To the new directory created in step 2.

4) Restart LabVIEW 2012 and create a new VI. Open the back-panel of the new VI and right-click anywhere. A new menu will appear. Navigate to User Libraries/BMCv5.0/ and drop each one of the three polymorphic VIs present in the library.
5) LabVIEW will not be able to find the ppcdll.dll needed for hardware access. Click the Browse button and point to ppcdll.dll located at C:\Program Files\BostonMicro\.
6) Drag the other two VIs into the back-panel and the remapping will automatically occur.
7) Close the new VI and save the three polymorphic VIs with the new DLL mapping.

You can now have access to all the BMC VIs directly from the LabVIEW menu.

VI) Hardware Reference Manual - Register Map:

This section of the User's Manual is for low level, advanced users. The user does not need to understand this section of the manual or directly access these registers if using the Matlab or LabView provided SDK. Higher level functions provide the user with the desired functionality without having to understand how the hardware operates. This section is for those who choose to develop stand-alone applications that interface directly to the DLL or the driver.

The PCIe HVA Interface board has a few simple 32-bit registers. All accesses must be 32-bits and fall on an even 32-bit address. When accessing the registers through the provided DLL, use the integer Register number. When accessing the card using custom code directly to the driver use the PCI offset address. Reserved registers are not to be accessed and are for future use. Some registers listed below remain for backwards compatibility and should not be used for new designs. Those functions are in italic print.

Reg Num	**PCI Offset**	**Access**	**Register Name**
0	0x00	Read Only	FPGA Hardware Revision
1	0x04	R/W	HVA Mode
2	0x08		*Reserved*
3	0x0C	Write Only	HVA Burst Abort
4	0x10	R/W	HVA Poke (31:16)=Act Num, (15:0)=Command
5	0x14	Read Only	Current HVA Position (15:0)=Position
6	0x18	R/W	HVA LUT
7	0x1C	R/W	DM Orientation (2:0)
8	0x20	R/W	Dither Period (16 ns counts)
9	0x24	R/W	Dither Waveform RAM (2048 maximum Length)
10	0x28	R/W	Dither Pattern
11	*0x2C*		*Reserved*
12	0x30	Read Only	Multi-X Status (4:1)
13	0x34	R/W	Test(31:0)
14	0x38		*Reserved*
15	0x3C	Read Only	Fiber Status
16	0x40	R/W	HVA Control
17	0x44	R/W	Sequence Period
18	0x48	R/W	Sequence Control
19	0x4C	R/W	Sequence Pattern
20	0x50	R/W	Sequence Delay
21	0x54	R/W	Test Address
22	0x58	R/W	Test Select
23	0x5C	R/W	TTL Stretch
24	0x60	Read Only	Kilo HVA Status
32-2079	0x80-0x207C	Write Only	Burst Window for all HVAs

Hardware Revision (0x00) – Register 0:

31:16	15:8	7:0
Design ID	Major Rev	Minor Rev

Default: 0x20000102

HVA Mode (0x04) – Register 1:

31:16	15:8	7	6	5	4:0
HVA Size(15:0)	Rsvd	Fiber Mode	Use Fiber	Use CLink	HVASel(4:0)

Default: 0x00000000

HVASel (4:0): These bits have no control over the hardware. Software can use these bits to signify a unique HVA/DM.

Use CLink: Set this bit to send the data to a Multi-X HVA. The maximum number of actuators the Camera Link port supports is 256.

Use Fiber: Set this bit to send the data to the Fiber Port. The maximum number of actuators the S-driver supports is 1024. The maximum number of actuators a Kilo Driver supports is 4096.

Fiber Mode: 0 = S-drive, 1 = Kilo Driver.

HVA Size (15:0): This is the size of the HVA/DM. The maximum size currently supported is 4096.

HVA Burst Abort (0x0C) – Register 3:

31	30:0
Abort	Reserved

Default: 0x00000000

Abort: Set this bit to end the current frame of Burst data. This can be used to end a Burst Frame. This should not be required unless the incorrect amount of data was send for the HVA size.

HVA Poke (0x10) – Register 4:

31:16	15:0
Actnum	Data

Default: 0x00000000

Software can write to any actuator on the HVA by using this HVA Poke register. Addressing follows the User Space addressing similar to Figures 2 or 3 and depends on the HVA mode.

Data (15:0): The HVA accepts unsigned (0-65535) counts.

ActNum (15:0): Use Figure 2 or 3 as an example to determine the desired actuator to poke. (Depending on the HVA Mode) Each write to this register will send a single DM command to the HVA.

Even the missing corners can be poked, although hardware will strip those commands out and not sent them to the HVA. Hardware will also strip out any erroneous actuator numbers. The maximum HVA size is currently 4096.

Current Mirror Position (0x14) – Register 5:

31	30:16	15:0
Rst	Reserved	Data

Default: 0x00000000

At any time software can read the current state of the mirror. This can be used for a user display or to check the status of the dither hardware.

Rst: Set this bit to reset the internal address of the RAM to zero.

Data (15:0): The current position of the internally addressed actuator.

There is no way to randomly read (peek) a single actuator on the DM. Software must reset, read the entire array of interest and then reset again to complete the readout sequence.

HVA LUT (0x18) – Register 6:

This register is one of the few registers that are not backwards compatible with previous versions of the software. HVADAC was from 7 down to 0 and Invalid was located at bit 8.

31	30:17	16	15:12	11:0
Rst	Reserved	Invalid	0000	HVADAC

Default: 0x00000000

This register is used to map the actuator address in user space to the individual DAC channels on the HVA. This is the last step before the commands leave the PCIe card. HVADAC addresses range from 0 to the size of the HVA.

Rst: Set this bit to reset the internal LUT address to zero. Each read or write access increments the address.

Invalid: Set this bit if the HVADAC channel does not exist for a particular user space channel, like the four corners.

HVADAC (11:0): This is the HVA DAC channel corresponding to a particular user space actuator number.

**Note that User space is defined as 1 to N, however the HVA is zero indexed. This is why HVA DAC addresses range from 0 to N-1.*

DM Orientation Mode (0x1C) – Register 7:

31:3	2:0
Reserved	Orien

Default: 0x00000000

These bits have no effect on the hardware, but software sets and reads them to determine the current orientation. The 8 possible orientations are listed below. When the DM orientation is changed in software, the rotated and flipped LUT is downloaded to the card each time.

Orien (2:0):

0	Normal
1	Flip X
2	Flip Y
3	Flip X and Y
4	Transpose
5	Transpose, Flip X
6	Transpose, Flip Y
7	Transpose, Flip X and Y

Dither Period (0x20) – Register 8:

31:0
Dither Period

Default: 0x00000000

Dither Period (31:0): Software writes a non-zero value to this register to begin dithering the mirror. This registers controls the rate in which the hardware will transmit a full 32 or 140 values to the HVA. A 62.5 MHz (16 ns) clock is used to synthesize this frame period. Set to zero to disable the dithering hardware. For example: If a 50 kHz frame rate is desired then set this register to:

1250*16ns = 20 µsec => 50 kHz.

The Dither waveform must be set up prior to enabling the dither hardware.

Dither Waveform (0x24) – Register 9:

31	30:17	16	15:0
Rst	Reserved	EOW	Data

Default: 0x00000000

Rst: Set this bit to reset the internal waveform address to zero. Each read or write access increments the address.

EOW: Set this bit WHILE writing the last value in the waveform to signify the end of the waveform. The internal address will reset back to zero and repeat the pattern stored in memory indefinitely. Waveform patterns can be any length between 2 elements (which will synthesize a square wave at twice the Dither Period) up to 2048 elements for more accurate waveforms.

Dither Waveform (15:0): The waveform can be any pattern between 0 and 1.0. The waveform is a fixed point unsigned 1.15 integer between 0 and 32768. For example: If the waveform element was 0.3333 then the actual value written to memory is:

0.3333 * 2^15 = 10922 which is not exactly represent and equals 0.33331298828125

Dither Pattern (0x28) – Register 10:

31	30:16	15:0
Rst	Reserved	ActCMD

Default: 0x00000000

Rst: Set this bit to reset the internal waveform address to zero. Each read or write access increments the address.

ActCMD (15:0): Software downloads a DMA pattern between 0 and 65535 for each actuator on the DM. The first write after a reset will be actuator 1 and each write afterwards will correspond to the next actuator address in User Space.

See Appendix I for more detailed information on dithering the mirror.

Multi-X Status (0x30) – Register 12:

31:5	3:0
Reserved	Status

Default: 0x0000000X

The four control signals (CC(4:1)) in the Camera Link® specification are utilized for limited HVA status.

Status (3): CC (4). Any Error has occurred since the HVA has powered up. This bit is a "sticky" bit and will stay set until the power is cycled. Possible errors include:

1) Writing to the same DAC too soon after a previous write.
2) Writing a "bad" address outside the acceptable range of addresses for that HVA.
3) Over Temp. Internal temperature is above 50 degrees Celsius.

Status (2): CC (3). DACs Busy. This bit is active whenever a write cycle (0.6 µsec) is in progress.

Status (1): CC (2). This bit is set upon when data is written to any location before the prior word can be read out of the same buffer and written to the respective DAC, causing this word to be overwritten and lost. The bit is reset when this latest word is read out and written to the DAC. For reliable detection, the host board should use edge triggered logic to detect a potentially very short pulse.

Status (0): CC (1). This bit is set upon a Camera Link® lock condition, as detected by the HVA FPGA, which enables the high voltage. This bit will go low if the link becomes unlocked, or in the case of an over-temp condition (internal air temp above 50° C) where the bit will be latched low until a power-cycle. Either condition will inhibit the high voltage.

Test Register (0x34) – Register 13:

31:0
Test Register

Default: 0x00000000

Test Register (31:0): Read/Write 32- bit register that is used for testing and has no effect on the hardware.

Fiber Status (0x3C) – Register 15:

31:7	6	5	4	3	2	1	0
Reserved	Hard ERR	Soft ERR	LN UP	CH UP	SFP LOS	SFP TXF	SFP nPRST

SFP nPRST: This bit is low if a SFP module is present in the cage.

SFP TXF: This bit is set if the SFP has a TX fault.

SFP LOS: This bit is set if the SFP has lost light in the fiber.

CH UP: This bit is set if the Fiber Link has successfully finished the startup handshaking protocol with the HVA.

LN UP: This bit is also set if the Fiber Link has successfully finished the startup handshaking protocol with the HVA.

Soft ERR: This bit is set if a periodic bit error has been received or if the Fiber Receiver decodes an invalid character.

Hard ERR: This bit is set if there is a hardware failure or loss of clock.

HVA Control (0x40) – Register 16:

This register provides a means for software to write directly to the Fiber port for either the S-driver or the KILO. Depending on the mode, this register takes on different forms.

S-Driver: This register provides direct access to either the 256 DACs on the HVA or a flat register map on the HVA FPGA. There are currently no registers on the S-driver.

31	30	29	28	27:16	15:0
Cntl	0	1	0	Address(11:0)	Data

Default: 0x00000000

Data (15:0): When accessing the DACs only Data (11:0) are valid.

Address (11:0): DAC access ranges from addresses 0x400-0x7FF. There are currently no registers on the HVA and all are reserved for future use.

Address Range	Use
0x000 – 0x3FF	*Reserved - HVA Registers TDB*
0x400 – 0x7FF	DAC access (Control & Data)

Bits (30:28) must always be “010”.

Cntl: This bit must be a 0 to access the DAC data registers. When set to a 1 the control registers on the DAC are accessed.

*Note: After the HVA1024 is powered, the 256 DACs are disabled and do not output any voltage. Every DAC channel, all 1024, must be enabled by accessing this register. This is done in the SetUpHVA functions. The following pseudo code enables all 1024 DACs.

```
for n = 1 : 1024 {
        Actnum = 1024+(n-1);
        WriteReg (16, 2^31+actnum*2^16+60);
        WriteReg (16, 2^31+2^30+actnum*2^16);
}
```

HVA Control (0x40) – Register 16:

This register provides a means for software to write directly to the Fiber port for either the HVA1024 or the KILO. Depending on the mode, this register takes on different forms.

KILO: This register provides direct access to the HVA DACs. It should only be used in factory testing.

31	30	29	28	27:16	15:0
0	DAV	0	0	Actnum(11:0)	Data

Default: 0x00000000

Data (15:0): Actuator Command.

Actnum (11:0): 0x000 – 0xFFF.

Sequence Period (0x44) – Register 17:

31:0
Sequence Period

Default: 0x00000000

Sequence Period (31:0): Software writes a non-zero value to this register to generate an onboard trigger to control the sequencing hardware. Sequencing will not begin until it is enabled in the Sequence Control register. This register controls the rate in which the hardware will transmit the next sequence of commands to the HVA. A 62.5 MHz (16 ns) clock is used to synthesize this frame period. Set to zero to disable the internal trigger and use the TTL input. For example: If a 50 kHz frame rate is desired then set this register to:

1250*16ns = 20 µsec => 50 kHz.

If this register is set to all zeros, the value read back will be the current period of the TTL input.

Sequence Control (0x48) – Register 18:

31	30:1	0
Rst	Reserved	Ena

Default: 0x00000000

Rst: Set this bit to reset the internal state machine. This should not be necessary unless the sequence pattern memory is not initialized properly. This register pulses internally and does not need to be cleared once set.

Ena: Set this bit to begin sequencing. If the Sequence period is zero then the TTL input is used to drive the frequency. If the Sequence period is non-zero then the internal trigger is used.

Sequence Pattern (0x4C) – Register 19:

31	30	29	28	27:16	15:0
Rst	0	EOP	EOF	Actnum (11:0)	Actcmd

Default: 0x00000000

Rst: Set this bit to reset the internal address to zero. Each read or write access increments the address.

EOP: End of Pattern. Set this bit while writing the last command to instruct the hardware to return to address zero and repeat the sequence. Both EOP and EOF should be set.

EOF: End of Frame. Set this bit while writing the last command in a frame to instruct the hardware to end the frame and wait for the next trigger.

Actnum (11:0): Each memory address can access the HVA in any random order. See below for more information.

Actcmd (15:0): The DAC command associated with Actnum.

*Note: The functions provided accept only full frames of images. (6x6, 12x12, or 32x32) The sequencing hardware however allows for random access and any size of frame. It is possible to develop a WriteSequenceCompress function that works in theory like video compression. Only the differences from one sequence pattern to the next are downloaded to the Sequence Pattern memory. This may reduce the number of values transmitted on each trigger and allow for more frames to be stored. The sequence pattern memory is 4096 in size.

Sequence Delay (0x50) – Register 20:

31:24	23:0
00000000	Sequence Delay

Default: 0x00000000

Sequence Delay (23:0): Software writes a non-zero value to this register to delay the sequencing hardware a period of time after the TTL trigger is received. Sequencing will not begin until it is enabled in the Sequence Control register. A 62.5 MHz (16 ns) clock is used to synthesize this delay. The delay can be set to close to 1/Frame period to trigger the sequencing just before the next trigger. For example: If a 10 µsecs delay is desired then set this register to:

1e-6 / 16e-9 = 62

Test Address (0x54) – Register 21:

31:29	27:16	15:0
0000	ActNum	ActCmd

Default: 0x00000000

ActNum (11:0): Set this to the desired actuator channel for the TTL to pulse when that actuator is written to the HVA.

ActCmd (15:0): Set this to the desired actuator command for the TTL to pulse when that command is written is written to the HVA

Test Select (0x58) – Register 22:

31	30:4	3:0
Stretch	zeros	Test Sel

Default: 0x00000000

Stretch: Set this bit to '1' to stretch the TTL pulse to a desired width controlled by the TTL Stretch register. If it is clear the TTL output will pulse for only 16 ns per occurrence.

Test Sel (3:0): These four bits control when the TTL output pulses.

0x0	Any command is sent to the HVA.
0x1	Any command to a specific actuator.
0x2	A specific command to any actuator.
0x3	A specific command to a specific actuator.
0x4	TTL output is high when FVAL is high.
0x5	Internal Error. (DAV is high without a FVAL)

TTL Stretch (0x5C) – Register 23:

31:16	15:0
zeros	Stretch Value

Default: 0x00000000

Stretch Value (15:0): When bit 31 is set in the Test Select register then this register controls how wide the TTL pulse will be for each occurrence. A 16 ns clock is used to control the pulse width. For example if a 10 µsec pulse is required set this register to:

10e-6/16e-9 = 625

Kilo Status (0x60) – Register 24:

31:3	2	1	0
Reserved	HF	ADDR OOR	DAC OVF

DAC OVF: Bit is set if the same DAC channel was written too quickly.

ADDR OOR: This bit is set if a bad address was received.

HF: The Kilo FIFO is Half Full

Burst Window (0x80-0x207C) – Register 32-2079

31:16	15:0
Command B	Command A

Command A (15:0): Odd HVA Actuator Command (1, 3, 5, etc.)
Command B (15:0): Odd HVA Actuator Command (2, 4, 6, etc.)

The FPGA has a window where any two actuators can be accessed in a single write. Bursting is the fastest way to transmit a frame of commands. Since all addresses must be on a 32-bit boundary, two actuators are always accessed when writing to this window. Depending on the HVA mode, when the correct number of commands has been transmitted, the internal frame line goes low signifying the end of a frame of actuator commands. In addition, setting the Abort bit in the HVA Burst Abort Register will also end a frame if for some reason software does not write a complete frame. The DM orientation can be used to flip or transpose the HVA commands.

Below lists the typical transfer times for various HVA/DM sizes using Window 7 and an inexpensive PC. Transfer times are not guaranteed. Windows 7 is not a real-time operating system.

HVA/DM	**Typical Transfer Time**
HVA32	445 ns
HVA140	1.78 µs
HVA492	10.5 µs – 11.4 µs
HVA952	15 µs – 16.4 µs
HVA1024	17 µs – 19.3 µs
HVA2040	35 µs – 37.3 µs
HVA4096	77 µs – 80 µs

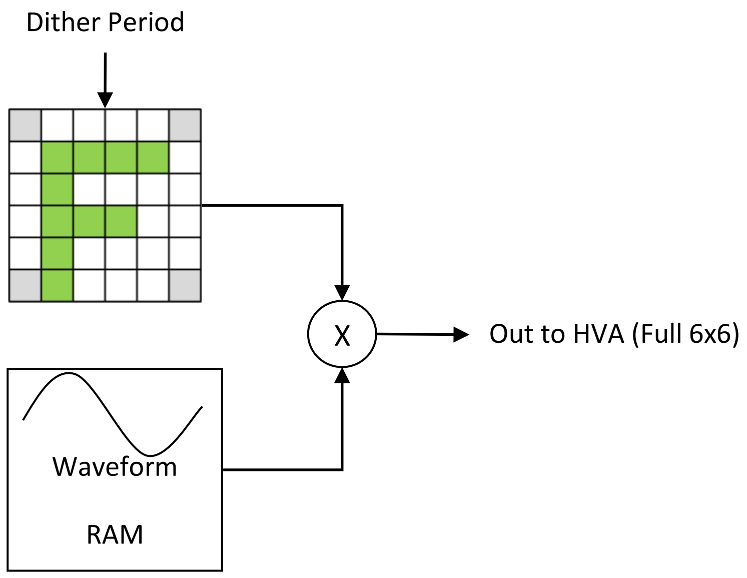
Appendix I: Dithering

The PCIe HVA interface board has the ability to generate full frames of data internally and transfer them rapidly to the HVA. Since this hardware does not utilize the PCIe bus to write commands, commands can be sent at rate limited only by the Camera Link® pixel clock or fiber rate. Note that dithering has the ability to transfer commands faster than the HVA can accept commands. An overflow condition occurs and will cause the CC (4) bit to set and latch.

Dithering requires three setup steps.

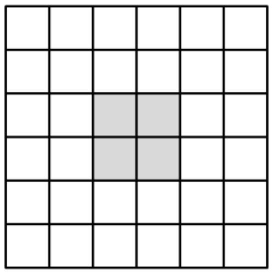
1) Download the Actuator Pattern. This becomes a fixed pattern, whose amplitude is adjusted in time. It represents the maximum command that will be reached during the dither process. This 2D pattern is multiplied by the current waveform element and sent to the HVA.
2) Download the Waveform. The waveform pattern can be between 2 and 2048 elements in length. A value is taken from the waveform, multiplied by all the values in the 2D actuator pattern, and transmitted to the HVA. The waveform is advanced to the next value and the process repeats at the desired rate until the last element in the waveform is reached. The address is reset back to the beginning and this repeats until disabled. The waveform can range from 0 to 1.0.
3) Set the Dither Frame Rate. Any non-zero value written to the Dither Period register will start the dither process. This rate IS NOT the outgoing oscillation rate seen on the mirror, but instead the rate in which the frames are transmitted to the mirror. The oscillation rate seen on the mirror is usually the Dither rate divided by the total length of the waveform. This is true only if the waveform contains a single cycle.

Figure 6: Dither Hardware.



Simple Dither Example:

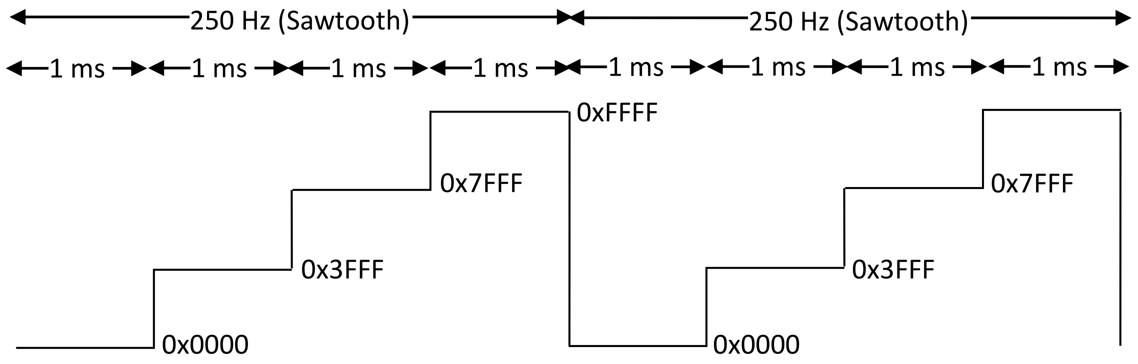
This is a simple dither example, a 2x2 center region of a 6x6 DM are dithered in a simple 4 element, relatively inaccurate, saw-tooth up to full stroke of the DM. The pattern sent to the Dither Pattern register is shown below along with the waveform seen on the four center channels. The Frame Period is set to 1 ms so the gray actuators in the center of the DM will oscillate at 250 Hz.



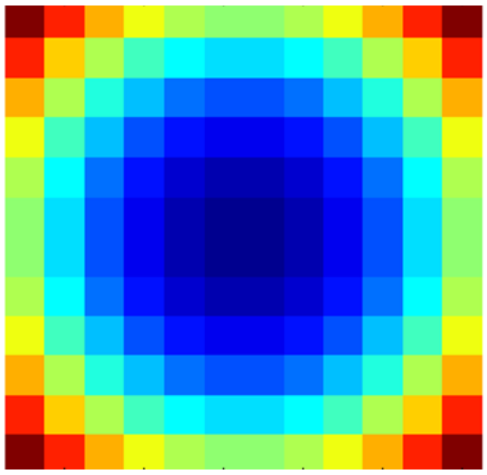
□ = 0x0000

▨ = 0xFFFF

Downloaded Waveform:
0.00
0.25
0.50
1.00



Note: The Dither Pattern does not need to be one value of the other. Each actuator can be any value. Even a Focus Pattern can be dithered. See Below.



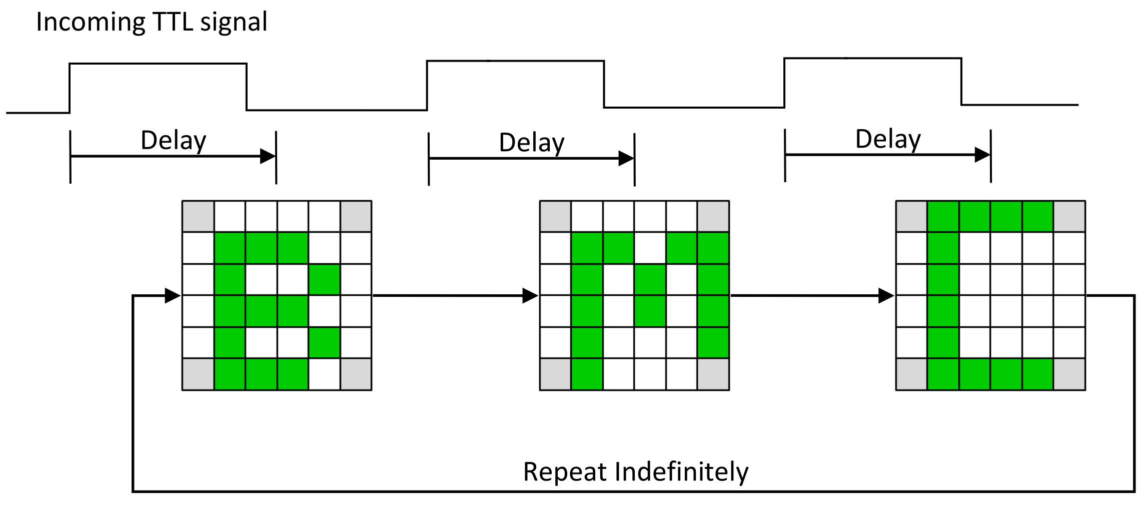
This 12x12 pattern has values ranging from 0x0000 (blue) to 0xFFFF (red).

Appendix II: Sequencing

The PCIe HVA interface board has the ability to generate full frames of data internally and transfer them rapidly to the HVA and sync this transfer to an external event. The TTL input is used to cycle through fixed DM patterns. Since this hardware does not utilize the PCIe bus to write commands, commands can be sent at rate limited only by the Camera Link® pixel clock or fiber rate. Note that sequencing has the ability to transfer commands faster than the HVA can accept commands. An overflow condition occurs and will cause the CC (4) bit to set and latch.

Sequencing requires three setup steps.

1) Download the DM Patterns. Software downloads a limited number of patterns to on board memory. The maximum number of patterns is determined by the size of the HVA.
2) Set the Delay. The transmission of the pattern to the HVA can be delayed a fixed amount of time. For example, this allows DM commands to be updated at the end of camera integration or readout, if the TTL input represents the camera integration or readout.
3) Enable the sequencing hardware. Software can choose to use the TTL input or use an onboard frame rate generator.



Appendix III: Creating Custom HVA/DM Combinations.

The PCIe HVA interface board supports any size HVA up to 4096 actuators. All the known HVA/DM combinations are described in the Matlab script Create_HVAs. (Labview equivalent is BMCHVA_config_v50.vi) If the user wishes to create a new HVA DM the following structure or cluster needs to be created for that HVA/DM combination. The Name of the Structure is the string the user inputs to SetUp the HVA. Add the custom HVA/DM to the script Create_HVAs.m and then run the script to add your HVA/DM to the .mat file. For Labview, constants have been created and are a part of the BMCHVA_config_v50.vi. Creating HVA can be time consuming and a very manual process. Especially with the HVA LUT.

Idnum: 0 to 31. Software can use this to signify which HVA/DM is currently active. This is necessary because the structure contains no dimension information. If the size is 144 the hardware does not know or care that the DM is 12x12. Idnum can be used to uniquely identify a 12x12 DM.

Use Camlink: This DM is connected to a Multi-X HVA.

Use Fiber: This DM uses the fiber port to either an S-driver or the Kilo driver.

Fiber Mode: 0 = S-driver, 1 = Kilo driver.

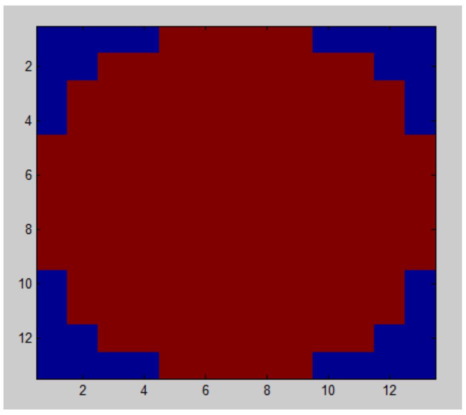
Burst Mode: Must be 0. Burst Mode 1 is currently not supported.

Size: This must be a 2-D number that represents the dimensions of the DM. Even if the DM can be thought of as a 1D array, the size must then be Nx1.

Inactive: This is an array of numbers that represent which actuators in the 2D dimension are not present. Indexes range from 1 to N*M. For example the HVA137 is 13x13 but has a round aperture. See the figure below.

Active Map: This is a 2D array of the same dimensions as the Size with ones and zeros representing the active and non-active region of the sub aperture.

HVA137 Active Map.



Hva Lut: This is a 2D array of the same dimensions as the Size with the DAC mapping. Values range from 0 to N-1. If there are many zeros in the LUT then the DM has several non-active actuators in the subaperture. Many HVAs have a default mapping because at the time the exact mapping was not known.

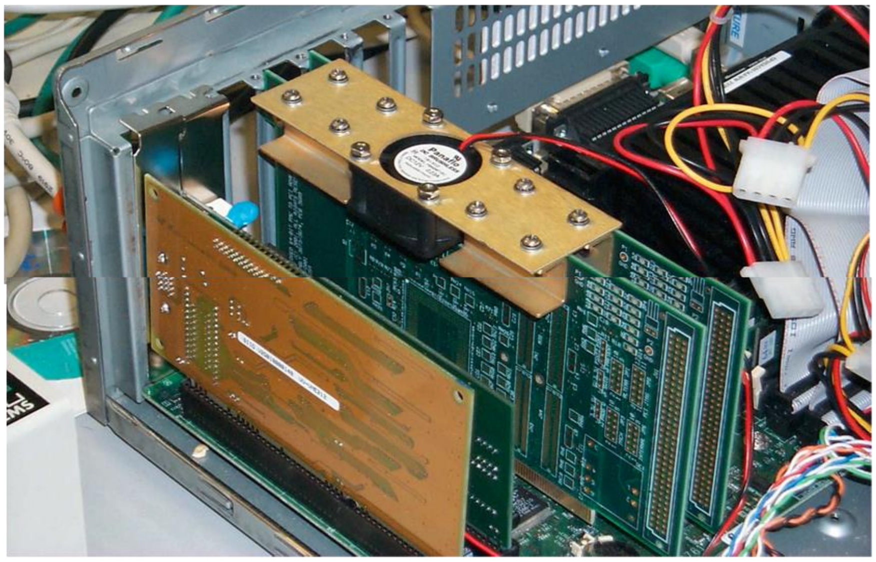
Once the custom HVA/DM has been created you may need to create a new Burst function in both Labview and Matlab. Use the existing functions as a guide. Make note of the special case if your HVA has an odd number of actuators. There may be a few places in the Labview that will also need updated to look for your new HVA string.

Appendix IV: AO System Integration Possibilities.

<u>Design A: Third Party Vendor System.</u>

The Boston Micro-Machine PCIe HVA Interface board owes its unusual shape for a very specific reason. It is intended to be paired with a XMC/PMC carrier from Technobox, Inc. In most cases, additional cooling is required, so the notch on the top of the board holds a fan assembly also available from Technobox. Technobox sells a variety of XMC/PMC carriers that can hold almost any 3rd vendor processing XMC/PMC. These XMC/PMC devices have the ability to interface to various sensors, do real-time AO processing, and send the data directly to the HVAIF board. XMC/PMC cards are a small mezzanine card that has connectors for either a PCI bus and/or a PCI Express® link. These cards are usually intended for a VME, VPX or Compact PCI chassis, but these Technobox adapters allow use in a standard PCI or PCI Express® slot. When paired with a Technobox XMC/PMC carrier, the HVAIF board produces a complete 2-slot AO system capable of high speed, low latency calculation (depending on the algorithm) and transmission of HVA commands.

Below is an image of two PCI or PCIe boards paired together with a fan assembly.



This table lists the components used in one possible AO system.

Vendor	Part Number	Description
Technobox	5243	PCIe 1X PMC Carrier
Technobox	4936	Fan Assembly
Curtiss Wright	XF05D-S05	XMC/PMC Processing Card
Curtiss Wright	CAML-MOD3	Dual Camera Link® Input Daughter Board.

The images below show various components used in this architecture.

Technobox Part Number: 5243 (PCIe 1X PMC Carrier)
http://www.technobox.com/pic5243.htm

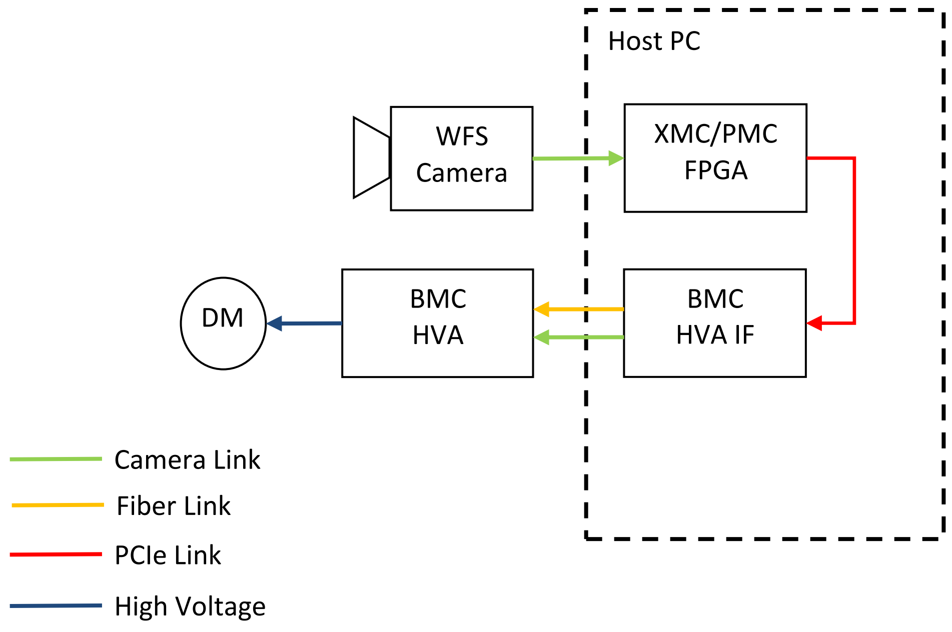
Technobox Part Number: 4936 (Fan Assembly for XMC/PMC Carriers)
http://www.technobox.com/pic4936.htm

Curtiss Wright Part Numbers:
XF05D-S05 (XMC/PMC FPGA Processing Board)
http://www.cwcembedded.com/pmc-fpga05d_fpga_processing.htm
CAML-MOD3 (Dual Camera Link® Daughter Card for XMC/PMC)
http://www.cwcembedded.com/caml-mod3_digital_io.htm

Alpha-Data Part Numbers:

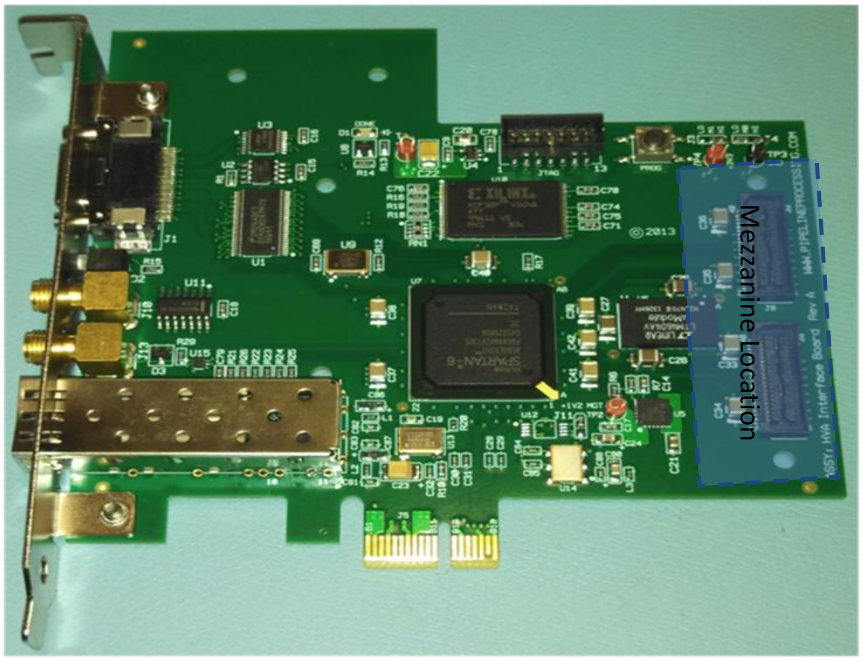
ADM-XRC-6T1/LV240T-1/Pn4 (XMC FPGA Processing Board)
http://www.alpha-data.com/products.php?product=adm-xrc-6t1
XRM-CLINK-MINI (Dual Camera Link® Daughter Card for XMC/PMC)
http://www.alpha-data.com/products.php?product=xrm-clink-mini
ADC-PCIE-XMC (PCIe, XMC Carrier)
http://www.alpha-data.com/products.php?product=adc-pcie-xmc

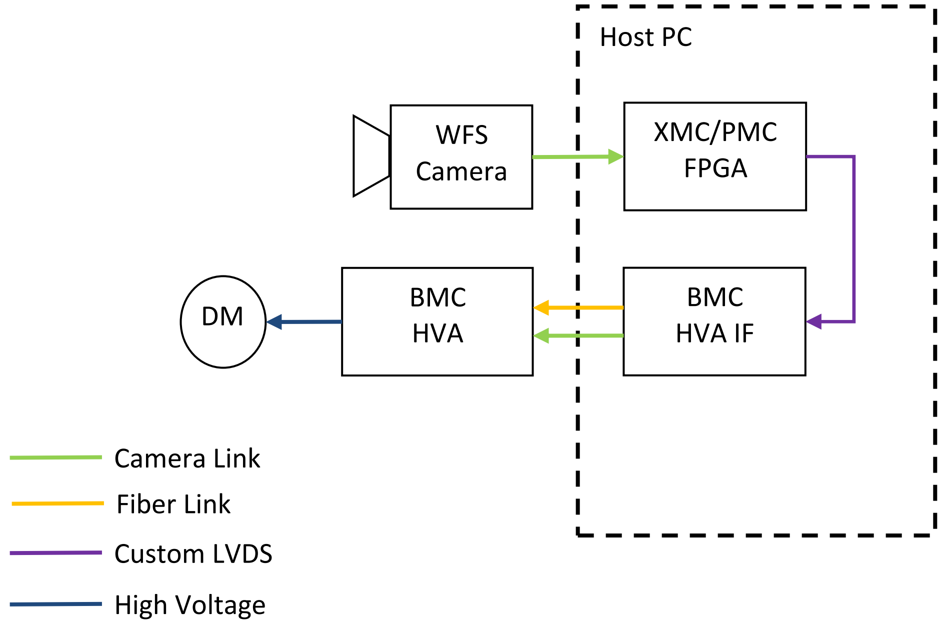
These few COTS components, along with a BMC HVAIF board and HVA/DM, create a powerful high speed AO system. The diagram below shows the data flow of this possible AO system.



Design B: Custom Solution with Mini-Mezzanine:

The Boston Micro-Machine PCIe HVA Interface board has two board-to-board connectors designed for use with a custom mezzanine. Any custom designed card will require a change to the Spartan FPGA firmware. The image below shows the PCIe board with the outline of a possible Mezzanine IO board. This board could be used to bring in HVA commands for a processing card via a simple LVDS (or other) interface. For more information on a custom Mini-Mezzanine send an email to info@pipelineprocessing.com





<u>Design D: Third Party Vendor System:</u>

A lower cost AO system is possible (depending on the algorithm) by using an FPGA based Camera Link® Frame-grabber. The AO algorithm could be implemented in the FPGA on the Frame-grabber and then send its results to the HVAIF board via the PCIe link. This cheaper alternative is also a 2-slot solution, but does not require the two boards to be inserted adjacent to each other in the host computer. Active cooling may also no longer be required. Below is a picture of an Inrevium prototyping frame-grabber.

Part Number: TB-6S-LX150T-GB-R
http://www.digilentinc.com