ACAM

COM Protocol

September 10 2021

Bruno Paillard

1 Introduction

The ACAM series of instruments can use a virtual Com port for communications and management of the instrument. The following document describes that Com protocol.

2 Com Enumeration

When the instrument is enumerated by the host PC, one of the interfaces that it presents is a virtual Com port (a CDC-Class USB device). On Windows 8.1 and up the generic Windows Com port driver is automatically instantiated and bound to that interface. On Windows 7, even though Microsoft provides a generic driver, the user must manually load the driver when the device is connected to the PC for the first time. After the driver is loaded, a new Com port is shown in the list of devices connected to the PC.

3 Com Configuration

The Com port can be configured (bit rate, number of stop bits… etc.), either using the controls in Windows *Device Manager*, or in an application by using the appropriate API functions. However such settings have **no effect** on the actual communication. They are only exposed for compatibility. At the hardware level there is no physical serial line present, and the ultimate communication speed is only determined by the throughput of the USB link. That throughput is typically a few Mb/s.

4 Communication Structure

Exchanges between the host PC and the instrument always follow a Master-Slave model. The host initiates an exchange using a *Command Packet*. The host may also send data following that *Command Packet*. The instrument responds either with the requested data, or with an Acknowledge if no data is to be transmitted back to the host.

In all cases after sending a command, the host PC must not send another command before the instrument sends a response back. That response may be data or may be an Ack if no data is requested by the command.

5 Endianness

Unless otherwise noted, the endianness is Big-Endian (MSB first).

6 Basic Types

The following basic types may be used in this protocol:

Type Name	Description	Endianness
U8	Single byte unsigned	N/A
U16	16-bit word unsigned	Big-Endian
U32	32-bit word unsigned	Big-Endian
U64	64-bit word unsigned	Big-Endian
I8	Single byte signed	N/A

I16	16-bit word signed	Big-Endian
I32	32-bit word signed	Big-Endian
I64	64-bit word signed	Big-Endian
Sgl	32-bit word in IEEE 754 floating point format	Big-Endian
Dbl	64-bit word in IEEE 754 floating point format	Big-Endian
String	Strings are concatenations of 8-bit ASCII characters, terminated by an end-of-text (0x00) byte.	N/A

Table 1

7 Packet Structure

7.1 Command Packet

The *Command Packet* is structured as follows:

Field	Size (bytes)	Function
Command	4	The command indicates the data transmitted or operation performed. Bit 31 of the command word indicates the direction of transfer: • 0: OUT (Host to Device) • 1: IN (Device to Host)
Address	4	The function of the address field varies with the command
Count	4	This field indicates the number of bytes to be transferred in the following data packet (either IN or OUT). How the bytes are interpreted is defined by the command. This number does not include the command packet.

Table 2

7.2 Data Packet

Data Packets are simply a concatenation of bytes. The way the bytes are interpreted is a function of the command that precedes the *Data packet*.

7.3 Acknowledge

The *Ack* is a single byte with value 0x06. The *Ack* byte is only sent back to the host if the command is a Write, and therefore does not require a data response from the device.

7.4 Commands

Command	Description	Address	Data/Ack
0x80000031	Read_Model	The address field is not relevant for this command	Data: ASCII string representing the Model. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000032	Read_SN	The address field is not relevant for this command	Data: ASCII string representing the serial number of the instrument. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000033	Read_FW_Rev	The address field is not relevant for this command	Data: ASCII string representing the Firmware revision. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000034	Read_FPGA_Rev	The address field is not relevant for this command	Data: ASCII string representing the FPGA Logic's revision.

			Size: Up to 32 bytes, including the termination byte Ack: No
0x80000035	Read_DOB	The address field is not relevant for this command	Data: U64 number representing the UTC (Universal Time Code) of the date/time of birth. The UTC represents the number of seconds elapsed since Jan 1 1904. Ack: No
0x80000036	Read_User_ID	The address field is not relevant for this command	Data: ASCII string representing the User-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte Ack: No
0x00000036	Write_User_ID	The address field is not relevant for this command	Data: ASCII string representing the user-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte

			Ack: Yes
0x800000A1	Read_Image This command retrieves all the pixels of the image	The address field is not relevant for this command	Data: N successive pixels. Each pixel is represented as 4-byte IEEE 754 float values. The MSB is sent first. The pixels are numbered from left to right and bottom to top. Pixel No 0 is at the bottom-left of the image. Ack: No
0x000000B1	Write_Stream_Index This command adjusts the beamformer's boresight to a pixel index. Effectively the signal that is streamed through the USB audio interface is the signal captured at the azimuth and elevation corresponding to the pixel index indicated.	The address field represents the pixel number corresponding to the beamformer's boresight for the audio signal that is streamed out to the host PC. Pixels are numbered from 0 to (N_{Pixel_rows} x N_{Pixel_Cols}) -1 from left to right and bottom to top. Pixel No 0 is at the bottom left of the image. Pixel 1 is just to the right of pixel 0 on the bottom row. The pixel just above pixel 0 in the image is numbered N_{Pixel_rows}.	Data: No Ack: Yes
0x000000B2	Write_Stream_Index_Dbg (Debug Mode) This command puts the instrument in debug mode, where the direct stream from a microphone is output on the I2S out channel. The address indicates the microphone number (from 0 to N_Channels-1)	The address field represents the microphone number for the audio signal that is streamed out to the host PC. Microphones are numbered from 0 to (N_{Sensor_rows} x N_{Sensor_Cols}) -1 from left to right and bottom to top. Microphone No 0 is at the bottom left of the array, when viewing the array from behind. Microphone 1 is just to the right of microphone 0 on the bottom row. The microphone just above microphone 0 in the image is numbered N_{Sensor_rows}.	Data: No Ack: Yes

0x000000C2	Write_Interpolation_Filter	The address field represents the number of coefficients transferred *N_Coefs*. This is typically 980 for an interpolation by 20 with 49 coefficients per interpolation filter.	Data: N*3 successive bytes. Each set of 3 bytes represents an 18-bit coefficient. Each coefficient is sent MSB first. The coefficients must be right-aligned. The contents of bit positions 18 to 23 (the first byte sent of each coefficient) does not matter. Ack: Yes
0x000000C3	Write_Persistence_Kt	The address field is not relevant for this command.	Data: I32 (4 bytes). This number represents an 18-bit value. That value is the time constant Kt of the lowpass image-persistence filter. The value is sent MSB first. The value must be right-aligned. The contents of bit positions 18 to 31 must be zero. Ack: Yes
0x800000D1	Read_Image_Parameters	The address field represents the type of parameter: **0: N_Array**	Data: 4 bytes representing

		The number of rows and columns of microphones in the antenna array The upper word represents the number of rows. The lower word represents the number of columns **1: N_Pixels** The number of rows and columns of pixels in the image The upper word represents the number of rows. The lower word represents the number of columns **2: I_Params:** - *N_Bits_per_Coef* The First byte represents the number of bits per coefficients (typically 18 for the latest implementation) - *N_Coefs_per_Interpolation* The next byte represents the number of coefficients per interpolation (typically 49 for the latest implementation) - *N_Bytes_per_Coef* The next byte represents the number of bytes per coefficient. This is typically 3 for 18-bit coefficients - *I_Factor* The last byte represents the interpolation factor (typically 20 for FOV 90 deg, or 32 for FOV 60 deg) **3: Fs:** - Returns 4 bytes that represent the sampling frequency in U32 format.	the requested data. Ack: No

Table 3

8 Annex 1 – Interpolation Filter Design

The interpolation filter's frequency response can be adjusted to limit the sensitivity of the camera to certain frequency bands.

Figure 1, Figure 2 and Figure 3 shows the default frequency and impulse responses of the filter for the 90 deg field of view. Figure 1 shows full frequency response of the filter. Figure 2 shows a zoom on the effective bandwidth of the camera. Figure 3 shows the impulse response of the filter. The filter is designed for a sampling frequency of 320 kHz. It has a 0 dB gain on most of the instrument's bandwidth, except near the Nyquist frequency, where it must decrease sharply.

The filter is an FIR filter that is typically linear phase (symmetrical). It must be designed with a predetermined number of coefficients, and for a predetermined interpolation factor. The default filter has 980 coefficients.

A new filter is written to the DSP engine of the camera by sending the successive samples of a new impulse response with the *Write_Interpolation_Filter* command. The filter new response is not persistent. It will revert back to the default filter response whenever the instrument is disconnected, or if the field of view is changed.

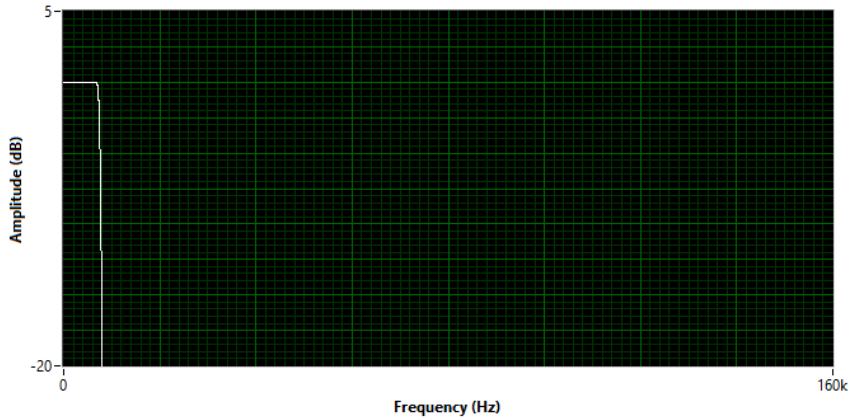

Figure 1

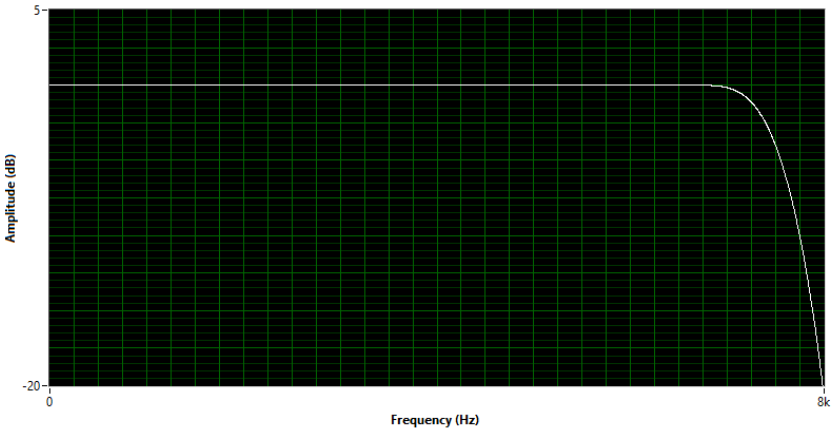

Figure 2

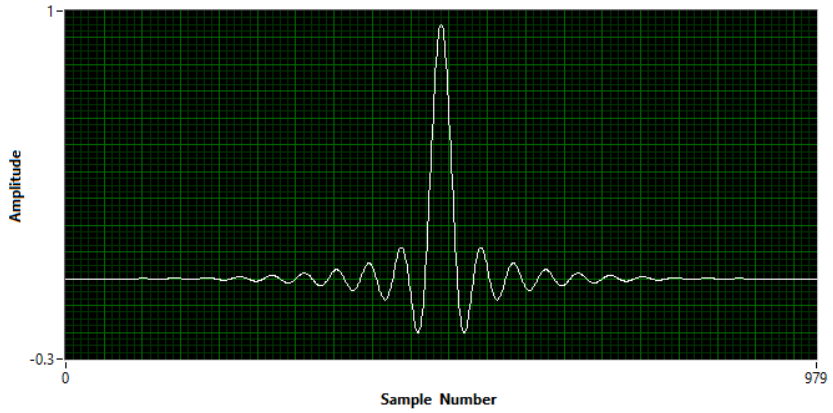

Figure 3

8.1 Filter Parameters

8.1.1 Image Process Parameters

In order to synthesize an appropriate filter, the first step is to find the process parameters that are in effect in the DSP engine of the camera. Different parameters can be in effect at any given time. For instance, changing the field of view of the camera, or firmware upgrade can affect these parameters. So, it is important to find those parameters before designing and writing a new filter response:

Image process parameters can be read using the *Read_Image_Parameters* command with the *I_Params* and *Fs* selectors.

The *I_Params* selector returns:

- *I_Factor*: The interpolation factor
- *N_Bytes_per_Coef*: The number of bytes per coefficient
- *N_Bits_per_Coef*: The number of bits per coefficient
- *N_Coefs_per_Factor*: The number of coefficients per interpolation factor.

The *Fs* selector returns:

- *Fs*: The base sampling frequency.

8.1.2 Filter Parameters

With that information, the filter can be designed with the following parameters:

- *N_Coefs*: The total number of coefficients of the filter must be equal to: $N\_Coefs = I\_Factor * N\_Coefs\_per\_Factor$
- *F_Filter*: The sampling frequency for which the filter is designed is calculated as: $F\_Filter = I\_Factor * Fs$
- *Fmax*: The effective bandwidth of the filter must be: $Fmax = Fs/2$. All the spectral shaping of the interpolation filter must occur between 0 and $Fmax$. Above $Fmax$ (between $Fmax$ and $F\_Filter/2$), the frequency response of the filter must be as low as possible to avoid any aliasing and insure a good camera performance without phantom images.

- *Dyn_Range*: The dynamic range of the coefficients is – 1 to +1. The representation is defined by the number of bits per coefficient. The coefficients are represented in fractional representation, with the binary point to the right of the MSB. This way coefficients between -1 and (almost) +1 can be represented. This representation is called $Q_{N\_Bits\_per\_Coef-1}$. In the current implementation, with *N_Bits_per_Coef* = 18, the representation is Q_{17}. In practice however, the definition of dynamic range is not essential. Let's assume that the filter that one wants to implement has coefficients between -4 and +4. One only must map that dynamic range to a Q_{17} representation. The amplitude of the image, and the audio signal streamed by the instrument, will be scaled accordingly.

8.1.3 Example

Let's say we want to modify the response of the camera, to be sensitive to components between 3.5 kHz and 7kHz.

We find the process parameters that are currently in effect in the camera. Let's assume we find the following:

- *I_Factor*: 20
- *N_Bytes_per_Coef*: 3
- *N_Bits_per_Coef*: 18
- *N_Coefs_per_Factor*: 49
- *Fs*: 16 kHz

From these values, we find the parameters of the filter:

- *N_Coefs*: 49x20 = 980
- *F_Filter*: 20x16000 = 320 kHz
- *Fmax*: Fs/2 = 8 kHz
- *Dyn_Range*: +- 1

The linear-phase filter is then synthesized using any appropriate filter synthesis tool, using these parameters. As well as the required bandwidth (3.5 kHz to 7 kHz).

Let's assume the filter synthesis tool produces the filter described in Figure 4 to Figure 6.

We then map the 980 filter coefficients to the Q_{17} representation. With that mapping, the largest coefficient has a binary value of 45141 (0x0B055).

We then use the *N_Bytes_per_Coef* = 3 value to segment each 18-bit coefficient in groups of 3 bytes (MSB first). With that segmentation the largest coefficient is represented as the three successive bytes 0x00, 0xB0, 0x 55.

We then use the *Write_Interpolation_Filter* command to send those 2940 bytes to the camera.

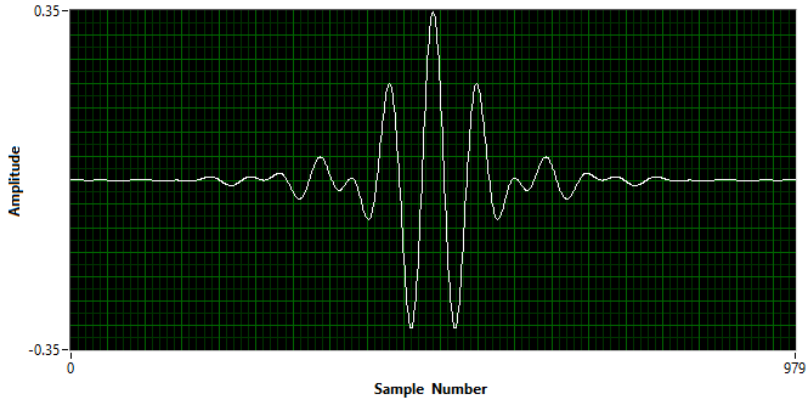

Figure 4

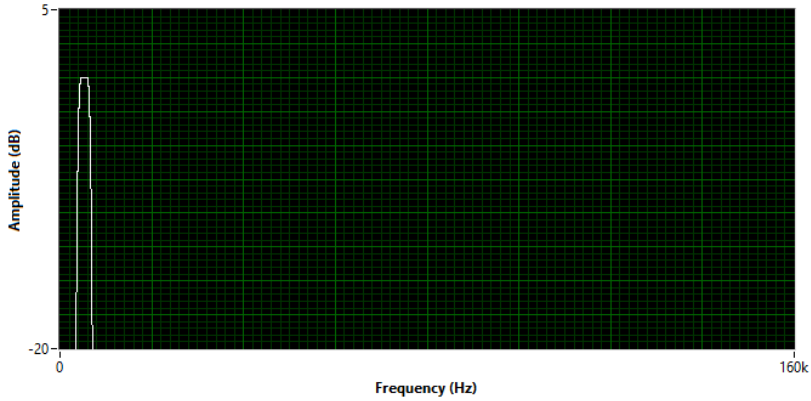

Figure 5

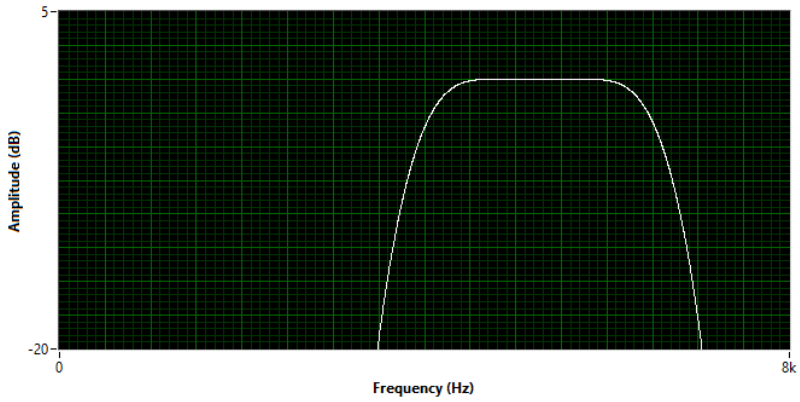

Figure 6

9 Annex 2 – Persistence Time Constant Representation

The persistence time constant K_t is represented as a positive fractional binary number on 18 bits (UQ_{18}). With that representation, the dynamic range of the K_t value is 0 to 1. K_t is sent to the camera as a 4-byte positive integer. Bits 18 to 31 of the value sent to the camera must be zero.

The relationship between the k_t value and the time constant of persistence τ (in seconds) is:

$$k_t = 1 - e^{\left(\frac{-1}{F_S * \tau}\right)}$$

So the longer the time constant τ, the smaller the k_t value.

The relationship between k_t and the binary value K_t is:

$$K_t = k_t * 2^{18}$$

So for instance, if F_S = 16 kHz, and we want to have a time constant of 0.5s, the binary value that must be transmitted would be K_t = 33 (0x00000021).

The following bytes must be sent using the *Write_Persistence_Kt* command:

0x00, 0x00, 0x00, 0x21