
Elab-080 DLL User's Guide

Table of Contents

Table of Contents	1
Introduction/Purpose.....	2
Using the DLL Driver	2
DLL Functions	3
GetHWStatus	3
GetSerialNumber	3
SetAnalogChannels.....	3
SetDSODCOffsets	4
SetDSOLASampleRate.....	4
CaptureDSOData	4
RetrieveDSOData	5
SetUPPSX.....	6
GetUPPSOverload	6
SetupAWG.....	7
AWGStart	9
AWGStop.....	9

Introduction/Purpose

This document's purpose is to describe the Windows DLL used to control and communicate with the Elab-080 hardware.

Using the DLL Driver

This DLL will automatically find the first Elab it can when it is called and will attach to that Elab. If more than one Elab is present, you can detect which Elab you are talking to using the GetSerialNumber command.

Once called, this DLL should be left open as it does have some internal memory of the status of the Elab hardware in order to minimize communication overhead.

DLL Functions

GetHWStatus

Prototype: bool GetHWStatus()

Usage:

This function checks to see if the ELAB-080 hardware is present and has power. This will return a 0 if the Elab HW is not found or if it does not have power. It will return a 1 if the Elab-080 is detected and has power.

This does not have to be called for the system to work correctly.

GetSerialNumber

Prototype: bool GetSerialNumber(long *SerialNumber)

Usage:

This function gets the serial number from the Elab. This function returns a “1” and the serial number in the “SerialNumber” pointer when the serial number is successfully read. When a failure occurs, “0” is returned in both fields.

Each time the DLL is called, it looks for the first Elab-080 it can find. If multiple Elab’s are plugged into a single PC, each can be accessed individually by running multiple instances of the DLL. This function can help you determine which Elab you are communicating with.

SetAnalogChannels

Prototype: int SetAnalogChannels(int CH1Gain, int CH2Gain, bool CH1Couple, bool CH2Couple, bool CH1Multiplier, bool CH2Multiplier)

Usage: Sets up the analog gain stages, DC/AC coupling, and 1X/10X probes.

Elements:

CHXGain: Possible values are 10, 20, 50, 100, 200, 500 mV/div. Send just the integer to set this value. If in 10X mode, the raw mV/div should be sent, but the results will come back at 100, 200, 500, 1000, 2000, or 5000 V/div since the probe does the division.

CHXCouple: Set to “1” for DC coupling, “0” for AC coupling. The capacitor in the Elab can take a few seconds to stabilize, so switching from DC to AC coupling requires a wait before the results will be correct.

CHXMultiplier: Set to “1” for 10X probes and “0” for 1X probes

Return: “0” if a gain setting isn’t an acceptable value, “1” for success

SetDSODCOffsets

Prototype: long SetDSODCOffsets(double CH1DCOffVoltage, double CH2DCOffVoltage)

Usage: Sets up the DC offset for each DSO channel

Elements:

CHXDCOffVoltage: Sets the DC offset as close as possible to the given voltage. This number represents the voltage that will now be the center of the ADC range, i.e. set to 2.5V if you wish for ground to be 2.5V below the center of the screen. The range of DC offset allowed is based on the current gain settings.

The range allowed is +/- (5*V/div setting). Thus in 5V/div, you can set to +/- 25V, in 20mV/div, this can be set to +/- 100mV.

Return: Sends “0” for success, a “1” if any channel cannot be set to the given voltage.

SetDSOLASampleRate

Prototype: double SetDSOLASampleRate(double DesiredFreq)

Usage: Set the sample rate for the DSO and LA channels

Elements:

DesiredFreq: The frequency, in Hz, that you would like the DSO and LA to sample at. Can range from 1,000Hz to 80MHz. Frequencies above 80MHz will program, and may work, but are not supported by Dynon.

Return: The actual frequency that was set.

CaptureDSOData

Prototype: int CaptureDSOData(unsigned int total_blocks, unsigned int pre_trigger_blocks, unsigned int trigger_channel, double trigger_value, unsigned int auto_roll, long OverrideOtherClocks)

Usage: Used to command the DSO and LA channels to take a capture, based on the current channel settings.

Elements:

total_blocks: The total number of 1K blocks to capture. Can range from 1-32.

pre_trigger_blocks: If the trigger is turned on, how many blocks will occur before the trigger. This ranges from 0 to (total_blocks – 1). One block must occur after the trigger.

trigger_channel: A value for what channel to trigger on. “0” for no trigger, ”1” for DSO CH1, “2” for DSO CH2, “3” for LA trigger.

trigger_value: This value is in volts if the trigger is a DSO trigger, and in bits if a LA trigger. The trigger must be “on screen”, which is +/- 5 divisions from the current DC offset. If the trigger is an LA trigger, only the least significant 4 bits are used, and one of these bits must be a “1”

auto_roll: If this is a “1,” the Elab will trigger every few seconds even if no trigger is found. This is sometimes helpful for finding a trigger. If this is a “0” the Elab will wait indefinitely until a real trigger is found.

OverrideClocks: Set to “0” if you wish the function to return a “0” if it cannot perform the clock settings. This is usually because the other clocks in the Elab are above or below 10KHz and you are trying to set these clocks in the opposite frequency band. Set this parameter to “1” if you wish for these clock settings to be programmed even if it corrupts other clocks in the Elab.

Return: “1” for success, “0” for failure

RetrieveDSOData

Prototype: long RetrieveDSOData(unsigned char whatchannels, double *DSOCH1, double *DSOCH2, unsigned short *LADATA, unsigned char Nth_Sample)

Usage:

This command retrieves any previously captured data from the Elab

Elements:

whatchannels: A bitmask of what channels will be active for the capture. The LSB is DSO CH1, bit 1 is DSO CH2, bit 2 is LA CH0-7, bit 3 is LA CH8-15. There is no advantage during the capture to less channels, but there is an advantage to upload speed.

*DSOCHX: A pointer to an array of voltage points from the DSO channels. This data returned is in fully calibrated volts.

*LADATA: A pointer to an array of LA data.

Nth_Sample: Allows the programmer to download only every Xth sample, speeding the upload. Usually used in continuous mode to speed the display. Set to “1” to download all data. Can range from 1-255. It is possible to do a “partial” download and then go back and do a “full” download as long as no capture is taken inbetween.

Return: The number of samples in the DSO and LA arrays. This number may not be (total blocks * 1024) as some samples at the beginning and end are thrown away for various purposes.

SetUPPSX

Prototype: float SetUPPS1(float Voltage)
 float SetUPPS2(float Voltage)

Usage:

This function sets the one of the User Programmable Power Supply channels to a given voltage. The range of voltages is from –12V to 12V. Since there are only 255 possible voltage settings, the actual voltage set is returned. The returned voltage is always rounded to one decimal point.

If the voltage is out of range, “9999” is returned. If the hardware is not present or not responding, “10000” is returned.

The Elab-080 is only specified from –10V to +10V, and dependent on the actual calibration of the Elab unit in use, reaching all the way to +12V or –12V may not be possible. Also, if the power supply is current limiting the output voltage may not be the voltage requested. Check “GetUPPSOverload” to see if the power supply is current limiting.

GetUPPSOverload

Prototype: int GetUPPSOverload()

Usage:

Checks to see if the User Programmable Power Supplies are current limiting or not. Returns “0” when no current limit, “1” when UPPS1 is limiting, “2” when UPPS2 is limiting, and “3” when both are limiting.

SetProgClocks

Prototype: double SetProgClocks (char Switch, double Frequency1, double Frequency2, double *Freq1Actual, double *Freq2Actual, long OverrideClocks)

Usage:

Sets the User Programmable Clocks to the given frequencies.

Elements:

Switch: Send “0” to turn off the programmable clocks, “1” to turn on.

FrequencyX: Send a frequency, in Hz, from 1000 to 150MHz to set channel X to the closest possible frequency

FreqXActual: Returns the actual frequency set. It’s not possible to hit every frequency requested exactly given the Elab’s hardware, so this result tells you what frequency was actually programmed. CH1 and CH2 interact with one another, so if you change the frequency on only one channel, expect the other to change from the previous actual frequency.

OverrideClocks: Set to “0” if you wish the function to return a “0” if it cannot perform the clock settings. This is usually because the other clocks in the Elab are above or below 10KHz and you are trying to set these clocks in the opposite frequency band. Set this parameter to “1” if you wish for these clock settings to be programmed even if it corrupts other clocks in the Elab.

Return: “0” for failure, “1” for success. All clocks in the Elab must be at or above 10KHz or below 10KHz at the same time. Failure will occur if you try and set one clock below 10KHz and the other above 10KHz, or if you have not set OverrideClocks and you attempt a setting that would violate this rule with respect to the DSO or AWG clocks.

SetupAWG

Prototype: long SetupAWG(double *AnalogVoltage, short *DigitalData, long BufferSize, double DCOffsetVoltage, double SampleFrequency, long Use4xGain, double OutputImpedance, long Repeat, long Triggered, long OverrideOtherClocks)

Usage:

Programs the AWG with the given settings. This function does not actually start the AWG.

Elements:

*AnalogVoltage: A pointer to an array of voltages for the Analog output of the AWG. These samples will be played in order and will jump from the last sample back to the first one in continuous mode. In 1X mode, this voltage can range +/- 1.25V, in 4X mode these voltages can range +/- 5V. The max output of the AWG into 50 ohms is

*DigitalData: A pointer to an array of digital data. Only 5 bits of digital output are available on the Elab, so only the five least significant bits of each array element will be used. All others will be discarded.

Buffersize: The number of samples to be played back. This can range from 10-65536.

DCOffsetVoltage: An analog DC offset to be applied to the analog waveform.

SampleFrequency: A frequency, in Hz for the playback rate of the AWG data.

Use4Xgain: Set to "0" for 1X mode, in which the step size is 2.5mV. Set to 10X mode to increase step size to 10mV.

OutputImpedance: The load that the AWG is driving. Since the AWG is a 50 ohm source, this needs to be set in order to correctly calculate the real output voltage of the AWG into a non-50 ohm load.

Repeat: Set to "1" to play the wave until "AWGStop" is called. Set to "0" to playback once and stop automatically.

Triggered: Set to "1" to only playback when the trigger input is a logic "1". Set to "0" to play back whenever AWGStart is called.

OverrideClocks: Set to "0" if you wish the function to return a "0" if it cannot perform the clock settings. This is usually because the other clocks in the Elab are above or below 10KHz and you are trying to set these clocks in the opposite frequency band. Set this parameter to "1" if you wish for these clock settings to be programmed even if it corrupts other clocks in the Elab.

Return: "0" for failure, "1" for success. All clocks in the Elab must be at or above 10KHz or below 10KHz at the same time. Failure will occur if you try and set one clock below 10KHz and the other above 10KHz and you have not set OverrideClocks.

AWGStart

Prototype: long AWGStart(long OverrideOtherClocks)

Usage:

Starts the AWG running with settings from SetupAWG. If the AWG is in single shot mode, this can be used to restart the AWG as many times as needed.

Elements:

OverrideClocks: Set to "0" if you wish the function to return a "0" if it cannot set the sample clock due to other clock settings. This is usually because the other clocks in the Elab are above or below 10KHz and you are trying to set these clocks in the opposite frequency band. Set this parameter to "1" if you wish for these clock settings to be programmed even if it corrupts other clocks in the Elab.

Return: "1" if successful, "0" if failure occurs.

AWGStop

Prototype: long AWGStop()

Usage:

Stops the AWG from running.

Elements:

Return: "1" if successful, "0" if failure occurs.