FASTDAQ REFERENCE MANUAL

Every command sent to the FastDAQ should be a string (ASCII characters). Every string should start with the operation to execute and end with the character that determines the end of the string, in this case ‘\r’ (carriage return).

General Syntax:

OPERATION,<data (varies with the operation)>’\r’(carriage return)

Where items enclosed in triangle brackets (<>) must be substituted for values

Where a DAC or ADC channel is specified, they are zero-indexed; An 8-channel DAC is addressed as channels 0-7, and a 4-channel ADC is addressed as channels 0-3.

The DAC-ADC AD5764-AD7734 (FastDAQ) can execute the following operations: “\*IDN?”, “\*RDY?”, “GET\_ADC”, “RAMP\_SMART”, “INT\_RAMP”, “SPEC\_ANA”, “CONVERT\_TIME”, “READ\_CONVERT\_TIME”, “GET\_DAC”, “ADC\_CH\_ZERO\_SC\_CAL”, “ADC\_CH\_FULL\_SC\_CAL”, “CAL\_ADC\_WITH\_DAC”, “WRITE\_ADC\_CAL”, “READ\_ADC\_CAL”, “DAC\_OFFSET\_ADJ”, “DAC\_GAIN\_ADJ”, “DAC\_RESET\_CAL”,“FULL\_SCALE”, “SET\_MODE”, “ARM\_SYNC”, “CHECK\_SYNC”, “ADD\_WAVE”, “CLR\_WAVE”, “CHECK\_WAVE”, “AWG\_RAMP”, “START\_PID”, “STOP\_PID”, “SET\_PID\_TUNE”, “SET\_PID\_SETP”, “SET\_PID\_LIMS”, “SET\_PID\_DIR”, “SET\_PID\_SLEW”. When the FastDAQ does not recognize the operation, it return the string “NOP”, which stands for “No Operation”.

A note about calibrations; The FastDAQ is pre-calibrated using a HP34401A DMM. Included with the Arduino Due code is a header file, ‘FastDAQcalconstants\_unitx.h’ from which the calibration settings are loaded on reset. Since the Arduino Due does not have EEPROM, the intention is that this file is renamed and recompiled for each new unit (we could also build an EEPROM into future units). The DAC channels should have a stable calibration, independent of various settings, largely eliminating the need for re-calibration in the short term. The ADC channels are pre-calibrated for the default conversion time of 395µs, and the calibration can change with different conversion times, especially for conversion times faster than ~300 µs. It is recommended to run the calibration routines, and record the calibration values, for the various conversion times that are intended to be used.

**\*IDN? and \*RDY?**

IDN? returns the string “DAC-ADC\_AD5764-AD7734\_*serialnumber*″.

Example:

\*IDN?

Returns:

DAC-ADC\_AD5764-AD7734\_UNIT2

RDY? returns the string “READY” when the DAC-ADC is ready for a new operation.

Example:

\*RDY?

Returns:

READY

**GET\_ADC**

GET\_ADC returns the voltage in mV read by an input ADC channel.

Syntax:

GET\_ADC,<adc channel>

Example:

GET\_ADC,0

Returns:

3.9999

**RAMP\_SMART**

Ramps one DAC channel \*\***in mV\*\*** to a specified setpoint at a given ramprate in 1ms steps. It looks up the current DAC value internally to make sure there are no sudden jumps in voltage. Internally it calls RAMP1 to do the actual ramp.

Syntax:

RAMP\_SMART,<dac\_channel>,<setpoint>,<ramprate>

Example:

RAMP\_SMART,3,4000,1000 (to ramp DAC3 to 4000mV at 1000mV/s)

Returns:

RAMP\_FINISHED

**INT\_RAMP**

INT\_RAMP ramps the specified DAC channels from the initial voltages to the final voltages and reads the specified ADC channels in a synchronized manner in a specified number of steps. It uses the ADC in a continuous sampling mode, and therefore there is no delay between updating the DAC output and acquiring the next ADC samples. While the ADC is acquiring the current samples, the next DAC step output values are being preloaded into the DAC, to be output synchronously as soon as the ADC samples are ready. The sample rate of this function is consistent, and capable of the maximum throughput of the ADC even while updating up to 8 DAC channels.

Oversampling, in order to do additional filtering by the control PC, is achieved by specifying a large number of steps (up to 232-1). Each DAC channel’s output value is treated as a 64-bit integer, and is scaled back to a 16-bit integer before being sent to the DAC. This allows a large number of samples to be taken without actually incrementing the 16-bit DAC output.

A ramp can be stopped at any time by sending the command “STOP” (without the quotes).

Syntax: (ALL mV)

INT\_RAMP,<dac channels>,<adc channels>,<initial dac voltage 1>,<…>,<initial dac voltage n>,<final dac voltage 1>,<…>,<final dac voltage n>,<# of steps>

**Do not** add commas between specified channels.

Example (ramping DAC channels 0, 6, and 7, while reading from ADC channels 0, 2, and 3):  
INT\_RAMP,067,023,-1000,-2000,-3000,3000,4000,5000,1000

Returns:

*<# of steps x number of selected adc channels x 16-bit integer samples>*RAMP\_FINISHED

**SPEC\_ANA**

SPEC\_ANA is similar to INT\_RAMP, but it does not ramp or change any DAC channels. The ADC channels to sample are specified, as well as the number of samples to take from each ADC channel.

An acquisition can be stopped at any time by sending the command “STOP” (without the quotes).

Syntax:

SPEC\_ANA, <adc channels>,<number of samples to take>

**Do not** add commas between specified channels.

Example (read from ADC channels 0, 2, and 3 for 3000 samples each):  
SPEC\_ANA,023,3000

Returns:

*<# of samples x number of selected adc channels x 16-bit integer samples>*RAMP\_FINISHED

**CONVERT\_TIME**

CONVERT\_TIME sets the conversion time, in µs, for each ADC channel, which is the time it takes to digitize the analog signal. The sum of the conversion times of all selected channels will determine overall sample rate. Shorter conversion times result in more measured noise; Refer to the AD7734 datasheet for typical noise vs conversion times (chopping is always enabled). For the AD7734, conversion times faster than approximately 300µs will start to exhibit a linear calibration offset >1mV at full range. If desired, this offset can be calibrated out using the provided calibration functions. Maximum conversion time: 2686µs. Minimum conversion time: 82µs. The function will return the actual closest possible setting.

Syntax:

CONVERT\_TIME,<adc channel>,<conversion time in µs>

Example:

CONVERT\_TIME,0,90

Returns:

82

**READ\_CONVERT\_TIME**

Reads the conversion time of the given ADC Channel in µs. See above (CONVERT\_TIME) for more info.

Syntax:

READ\_CONVERT\_TIME, <adc channel>

Example:

READ\_CONVERT\_TIME, 3

Returns:

82

**GET\_DAC**

GET\_DAC returns the current setpoint (in mV) of the specified DAC channel.

Syntax:

GET\_DAC,<dac channel>

Example:

GET\_DAC,1

Returns:

0.6104

**ADC\_CH\_ZERO\_SC\_CAL**

ADC\_CH\_ZERO\_SC\_CAL performs a system zero calibration on the specified ADC channel, where the input of the selected ADC channel should be held at 0VDC. This will calibrate out any offsets in the system due to the ADC itself, or any signal conditioning on the input. The function will return the calibration constant that has been stored to the ADC system zero-scale cal register, which can be noted or saved for restoring a calibration later. It is preferable to repeat this calibration any time the conversion time is changed using CONVERT\_TIME, especially when using conversion times less than 300µs.

Syntax:

ADC\_CH\_ZERO\_SC\_CAL,<adc channel>

Example:

ADC\_CH\_ZERO\_SC\_CAL,1

Returns:

ch<adc channel>,<zeroscale>

CALIBRATION\_FINISHED

**ADC\_CH\_FULL\_SC\_CAL**

ADC\_CH\_FULL\_SC\_CAL performs a system full-scale calibration on the specified ADC channel, where the input of the selected ADC channel should be held at positive full-scale (typically 10VDC). This will calibrate out gain errors in the system due to the ADC itself, or any signal conditioning on the input. The function will return the calibration constant that has been stored to the ADC system full-scale cal register, which can be noted or saved for restoring a calibration later. It is preferable to repeat this calibration any time the conversion time is changed using CONVERT\_TIME, especially when using conversion times less than 300µs.

Syntax:

ADC\_CH\_FULL\_SC\_CAL,<adc channel>

Example:

ADC\_CH\_ZERO\_SC\_CAL,1

Returns:

ch<adc channel>,<fullscale>

CALIBRATION\_FINISHED

**CAL\_ADC\_WITH\_DAC**

CAL\_ADC\_WITH\_DAC makes use of ADC\_CH\_ZERO\_SCALE\_CAL, ADC\_CH\_FULL\_SCALE\_CAL, and SET functions to calibrate ADC channels 0-3 using DAC channels 0-3. Before running the command, DAC channels 0-3 should be connected to ADC channels 0-3 using BNC cables. The function will automatically set the DACs to 0VDC and then 10VDC during the calibration cycle. It will return the zero-scale and full-scale calibration registers for each ADC channel.

Syntax:

CAL\_ADC\_WITH\_DAC

Example:

CAL\_ADC\_WITH\_DAC

Returns:

ch0,<zeroScale ch0>,ch1,<zeroScale ch1>,ch2,<zeroScale ch2>,ch3,<zeroScale ch3>,ch0,<fullScale ch0>,ch1,<fullScale ch1>,ch2,<fullScale ch2>,ch3,<fullScale ch3>,

CALIBRATION\_FINISHED

**WRITE\_ADC\_CAL**

WRITE\_ADC\_CAL is used to restore previously noted or saved ADC per-channel calibration register values, for both the zero and full scale calibration points.

Syntax:

WRITE\_ADC\_CAL,<adc channel>,<zero scale cal value>,<full scale cal value>

Example:

WRITE\_ADC\_CAL,1,826,2100756

Returns:

CALIBRATION\_CHANGED

**READ\_ADC\_CAL**

READ\_ADC\_CAL is used to display the currently loaded per-channel ADC calibration register values, for both the zero and full scale calibration points.

Syntax:

READ\_ADC\_CAL,<adc channel>

Example:

READ\_ADC\_CAL,1

Returns:

ch<adc channel>,<zeroscale>ch<adc channel>,<fullscale>

READ\_FINISHED

**DAC\_OFFSET\_ADJ**

DAC\_OFFSET\_ADJ uses the AD5764 DAC’s internal registers to remove any zero offset for each channel. The DAC provides a register per channel, which allow zero adjustment in increments of 1/8 of an LSB (38µV for a +/-10V range) in the range of -32 to +31 LSBs. Before performing a calibration, DAC\_RESET\_CAL should be used to zero the calibration registers. To perform the calibration, a reference DMM is connected to the DAC channel to be calibrated. With the DAC channel set at 0V, the DMM reading is noted and input into the function, which will automatically calculate and return the appropriate register values.

Syntax:

DAC\_OFFSET\_ADJ,<dac channel>,<dmm zero reading>

Example (with a DMM reading of -0.000070):

DAC\_OFFSET\_ADJ,0,-0.000070

Returns:

ch<dac channel>,<offset stepsize>,<offset register>

CALIBRATION\_FINISHED

**DAC\_GAIN\_ADJ**

DAC\_GAIN\_ADJ uses the AD5764 DAC’s internal registers to remove any gain error for each channel. The DAC provides a register per channel, which allow gain adjustment in increments of 1/2 of an LSB (153µV for a +/-10V range) in the range of -32 to +31 LSBs. Before performing a calibration, DAC\_RESET\_CAL should be used to zero the calibration registers, and the DAC\_OFFSET\_ADJ calibration performed first. To perform the gain calibration, a reference DMM is connected to the DAC channel to be calibrated. With the DAC channel set at negative scale (-10VDC), the DMM reading is noted and the calculated error is input into the function, which will automatically calculate and return the appropriate register values.

Syntax:

DAC\_GAIN\_ADJ,<dac channel>,<dmm error from negative full scale>

Example (with DMM reading of -9.9969V):

DAC\_GAIN\_ADJ,0,0.00310

Returns:

<gain stepsize>,<gain register>

CALIBRATION\_FINISHED

**DAC\_RESET\_CAL**

DAC\_RESET\_CAL is used to zero the calibration registers for a specified channel. This should be used before attempting to recalibrate a DAC channel with a reference DMM.

Syntax:

DAC\_RESET\_CAL,<dac channel>

Example:

DAC\_RESET\_CAL,0

Returns:

CALIBRATION\_RESET

**FULL\_SCALE**

FULL\_SCALE can be used to set the full-scale range of the DAC outputs to something other than the default 10V. This could be useful if a voltage divider or external circuit is used to alter the range of the DACs and you want to have this scale factored in automatically.

Syntax:

FULL\_SCALE,<DAC positive full-scale range in Volts>

Example (Setting the full scale range to 5V):

FULL\_SCALE,5.0

Returns:

FULL\_SCALE\_UPDATED

# MASTER/SLAVE FUNCTIONS

As of June 2020, new units have optical clock and sync inputs and outputs to allow synchronization between units. Functions that support sync are SPEC\_ANA, INT\_RAMP, and AWG\_RAMP. Each unit must have the same number of ADC channels being sampled, and the conversion time must be the same. One unit must be set to MASTER, all others to SLAVE. Units can be daisy-chained with the MASTER unit’s CLK\_OUT and SYNC\_OUT connected to the first SLAVE unit’s CLK\_IN and SYNC\_IN, and so on for each additional slave. The general sequence would be:

1. PC uses SET\_MODE to select one unit as MASTER, and all others as SLAVE. It can take up to 20 seconds for the PLL to relock switching between MASTER and SLAVE if the clock frequency is substantially different, which can be checked with CHECK\_SYNC
2. PC sends ARM\_SYNC to MASTER
3. PC sends INT\_RAMP (for example) to each SLAVE unit, which will wait for the SYNC signal from MASTER
4. PC sends INT\_RAMP to MASTER unit.
5. PC waits for RAMP\_FINISHED from each unit
6. The process can be repeated from step 2 (ARM\_SYNC)

**SET\_MODE**

SET\_MODE is used to select the INDEP/MASTER/SLAVE mode for each unit, valid modes are MASTER, SLAVE, and INDEP. When in INDEP mode, the unit will not check or wait for a valid SYNC signal before starting a sequence but still checks the CLK. The default mode on power-up is INDEP.

Syntax:

SET\_MODE,<mode>

Example:

SET\_MODE,INDEP

Returns:

INDEP\_SET

**ARM\_SYNC**

ARM\_SYNC is sent from the PC to the MASTER, before sending the sampling command (ie. INT\_RAMP) to each SLAVE, and lastly to the MASTER. The PC should verify that all units have finished the previous sequence before sending ARM\_SYNC. If the mode is INDEP, or SLAVE, this command is not necessary.

Syntax:

ARM\_SYNC

Example:

ARM\_SYNC

Returns:

SYNC\_ARMED

**CHECK\_SYNC**

CHECK\_SYNC is used to determine the state of the clock PLL, and the SYNC signal. Possible returned values are:

CLOCK\_NOT\_READY - The clock PLL is still trying to lock to the input clock signal

SYNC\_NOT\_READY - This message will only appear if in MASTER or SLAVE mode. For the MASTER this means ARM\_SYNC still needs to be called, for a SLAVE it could potentially be a bad or incorrect optical SYNC signal, or ARM\_SYNC still need to be called on the MASTER.

CLOCK\_SYNC\_OK – Both signals are acceptable for the current mode

Syntax:

CHECK\_SYNC

Example:

CHECK\_SYNC

Returns:

CLOCK\_NOT\_READY  
SYNC\_NOT\_READY

# AWG FUNCTIONS

The FastDAQ has an arbitrary waveform mode where a sequence of DAC setpoints can be assigned to 2 separate wave arrays. Up to 100 setpoints can be configured for each wave, and the number of ADC samples to take at each setpoint is specified. When an AWG\_RAMP sequence is started, the AWG DAC channels assigned to each wave loop through the sequence a specified number of times, then the ramp DAC channels will take a step. The AWG waveforms will repeat until the ramp DAC channels finish their ramp.

**ADD\_WAVE**

ADD\_WAVE is used to configure the arbitrary DAC setpoints. For each setpoint the number of ADC samples to take is also specified (up to 232-1). The function can be called multiple times until up to 100 setpoints have been stored, and should be sent in groups of 20 setpoints or less to avoid a possible serial buffer overflow.

Syntax:

ADD\_WAVE,<wave number (0 or 1)>,<Setpoint 0 in mV>,<Number of ADC samples to take at setpoint 0>,…<Setpoint n in mV>,<Number of samples to take at Setpoint n>

Example (setting 4 setpoints with various sample lengths to wave 0):

ADD\_WAVE,0,100.0,50,500.0,25,200.0,100,-5000.0,25

Returns:

WAVE,0,4

**CHECK\_WAVE**

CHECK\_WAVE returns how many setpoints, and the total number of samples, have been configured for the specified wave number (0 or 1)

Syntax:

CHECK\_WAVE,<wave number>

Example (for waveform in example above):

CHECK\_WAVE,0

Returns:

WAVE,0,4,200

**CLR\_WAVE**

CLR\_WAVE resets the number of configured setpoints for a specified waveform back to 0

Syntax:

CLR\_WAVE,<wave number>

Example:

CLR\_WAVE,1

Returns:

WAVE,1,0

**AWG\_RAMP**

Similar to INT\_RAMP you specify which ADC channels to sample and DAC channels to ramp, as well as number of ramp steps. Additionally, you select the number of independent waveforms (currently max 2), The DAC channels assigned to each waveform, and the number of waveform repetitions at each ramp step. If the waveforms are different lengths, the repetition counter will increment when any waveform completes.

A ramp can be stopped at any time by sending the command “STOP” (without the quotes).

Syntax:

AWG\_RAMP,<number of independent waveforms>,<DAC channels assigned to waveform 0>,<…>,<DAC channels assigned to waveform N>,<DACs to ramp>,<ADCs to sample>,<Initial DAC voltage 1>,<…>,<Initial DAC voltage N>,<Final DAC voltage 1>,<…>,<Final DAC voltage N>,<# of waveform repetitions at each ramp step>,<# of ramp steps>

Example (Use 1 waveform, assign DAC 7 to waveform 0, Ramp DACs 1 and 3, Sample ADC 0, Start DAC 1 at -5V, Start DAC 3 at -2.5V, Finish DAC 1 at 5V, Finish DAC 3 at 2.5V, Repeat waveform 10 times at each ramp step, 100 ramp steps):

AWG\_RAMP,1,7,13,0,-5000,-2500,5000,2500,10,100

Returns:

*<# of steps \* number of samples in wave \* number of repetitions \* number of selected adc channels \* 16-bit integer samples>*RAMP\_FINISHED

# PID FUNCTIONS

Somewhat experimental, the FastDAQ can act as a PID controller, currently only ADC 0 acts as the input, and DAC 0 acts as the output. While in PID mode, no other commands which use the ADC should be issued (INT\_RAMP, SPEC\_ANA, AWG\_RAMP, GET\_ADC…), but DAC ramping channels other than DAC 0 via RAMP\_SMART should be OK. While in PID mode, the loops runs continuously at the selected ADC 0 conversion time, and every 10th sample from ADC 0 and the output setting of DAC 0 will be returned as little-endian 4-byte floats followed by framing sequence 0xA5 0x5A. This formatting and framing sequence was chosen for easy testing and live tuning with RealTerm 3.0.1.44 and could be changed. The PID routines are based on this library with some modifications:

<https://github.com/br3ttb/Arduino-PID-Library>

All PID functions can be called live while the PID is running or before it is started.

**START\_PID**

START\_PID starts the PID controller, and the stream of samples from ADC 0

Syntax:

START\_PID

Example:

START\_PID

Returns:

Stream of:

<4-byte float ADC0 in mV><4 byte float DAC0 in mV><0xA5><0x5A>

**STOP\_PID**

STOP\_PID stops the PID controller and the stream of samples from ADC 0

Syntax:

STOP\_PID

Example:

STOP\_PID

Returns:

Nothing

**SET\_PID\_TUNE**

SET\_PID\_TUNE allows setting the proportional, integral, and derivative coefficients of the PID controller

Syntax:

SET\_PID\_TUNE,<P-coeff>,<I-coeff>,<D-coeff>

Example:

SET\_PID\_TUNE,1.0,0.25,0.0

Returns:

Nothing

**SET\_PID\_SETP**

SET\_PID\_SETP sets the value in mV that the controller should try to achieve at the input ADC 0

Syntax:

SET\_PID\_SETP,<Setpoint in mV>

Example:

SET\_PID\_SETP,500.0

Returns:

Nothing

**SET\_PID\_LIMS**

SET\_PID\_LIMS sets the minimum and maximum values in mV that the DAC 0 output should not exceed

Syntax:

SET\_PID\_LIMS,<Minimum output in mV>,<Maximum output in mV>

Example:

SET\_PID\_LIMS,-500,250

Returns:

Nothing

**SET\_PID\_DIR**

SET\_PID\_DIR defines the polarity for the output of the PID controller

Syntax:

SET\_PID\_DIR,<0 for Reverse, 1 for Forward>

Example:

SET\_PID\_DIR,1

Returns:

Nothing

**SET\_PID\_SLEW**

SET\_PID\_SLEW configures the slew rate limit for the output that will be enforced regardless of the calculated output from the controller

Syntax:

SET\_PID\_SLEW,<Maximum slew rate in mV per second>

Example:

SET\_PID\_SLEW,50.0

Returns:

Nothing