# A2.4.5 - TWM structure

This report covers also:

A2.1.1 – Flow chart of TWM tool

A2.1.2 – Extension for a multiple digitizers

A2.1.4 – Concept of the LV to Octave/Matlab interface

A2.2.2 – Integration of the drivers to the virtual driver

A2.2.4 – Control and acquisition modules

A2.4.1 – Building the TWM tool (LabVIEW)

A2.4.2 – TWM tool structure

A2.4.3 – Acquisition and control module description

A2.4.4 – Processing module description

A3.3.3 – Guidance on integration of new HW

Following text describes internal structure of the TWM (LabVIEW version).

## References

1. TWM tool, url: <https://github.com/smaslan/TWM>
2. INFO-STRINGS, url: <https://github.com/KaeroDot/info-strings>
3. QWTB toolbox, url: <https://qwtb.github.io/qwtb/>
4. GOLPI interface, url: <https://github.com/KaeroDot/GOLPI>
5. A232 Algorithms exchange format, url:

[https://github.com/smaslan/TWM/tree/master/doc/A232 Algorithm Exchange Format.docx](https://github.com/smaslan/TWM/tree/master/doc/A232%20Algorithm%20Exchange%20Format.docx)

1. A231 Correction Files Reference Manual, url:

<https://github.com/smaslan/TWM/tree/master/doc/A231 Correction Files Reference Manual.docx>

1. A231 Data Exchange Format, url:

[https://github.com/smaslan/TWM/tree/master/doc/A231 Data exchange format and file formats.docx](https://github.com/smaslan/TWM/tree/master/doc/A231%20Data%20exchange%20format%20and%20file%20formats.docx)

## Abbreviations

LV – LabVIEW

CVI – LabWindows CVI

EOS – End of string

DWORD – unsigned 32bit variable

INT16 – signed 16bit integer

INT32 – signed 32bit integer

INT64 – signed 32bit integer

Double – 64bit real number

Cluster – LabVIEW structure of elements

Bool – Logic variable

HDD – Hard drive

TWM – The LV program developed in scope of TracePQM project

GUI – Graphical User Interface

HW – HardWare

QWTB – Q-Wave toolbox [3]

INFO – Brain-dead structured, human readable text file

Matlab – Matlab SW (Mathworks)

GNU Octave – Open source equivalent of Matlab that happens to be almost 100% comatible

m-script – Matlab/Octave’s function file

## Overview

The TWM is organized according to the diagram shown in Figure 0‑1. The whole TWM application consists of two parts:

1. LabVIEW modules (Control and Processing) that controls the instruments, initiates processing and serves as a user interface
2. Calculation or Processing module based on the Matlab/GNU Octave which performs the processing of the acquired data, post-processing and formatting the data for displaying and generation of the measurement report (summary of the results formatted in compact form).

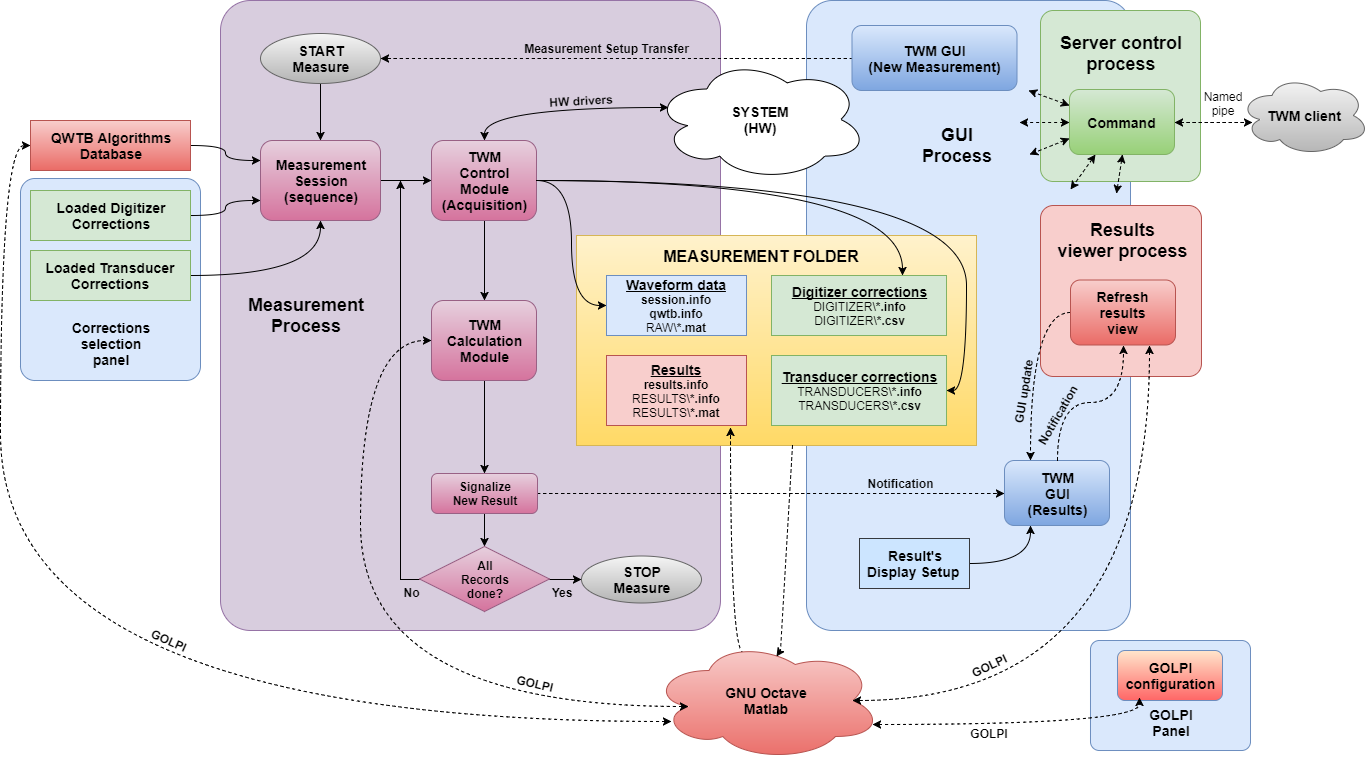


Figure 0‑1: TWM tool structure. The coloured frames are used to distinguish the process in which the tasks run.

The modules communicate on runtime via the GOLPI interface [4] and via files in the measurement folder [7]. So TWM appears as one interactive application. This apparent complication has several benefits. The separation of the acquisition and processing enables several features:

1. The acquired data may be processed at any time. It is possible to just record batch of measurements without processing which may be helpful for time consuming calculations. The processing of the whole batch of measurements can be initiated later either via TWM or on a supercomputer.
2. The same acquired data can be used for calculation of multiple parameters using multiple algorithms.
3. The measurement data is (can be) archived so the data may be reprocessed later if new parameters or correction are needed.
4. The Processing module can run independently on the Control module so TWM can run even without installed Matlab/GNU Octave and the processing can run on any system without drivers required for the TWM (e.g. supercomputer).
5. The Processing module is identical for LabVIEW and CVI version of the tool and the data are interchangeable.
6. The processing module is FULLY transparent. The m-functions of the module do everything: loading the acquired data, loading correction, processing, saving results, loading and formatting results for display, generating report.

The control module is split into four separate processes that run in parallel. Main process is “**GUI Process**”. It contains configuration panels of the HW, configuration panels of the measurement, configurations of the result display and selector of the correction files for the HW components.

When the user wants to initiate a new measurement the “**GUI process**” will create “**Measurement Process**” which does following:

1. Loads correction files.
2. Loads selected QWTB algorithm’s configuration from QWTB alg. database file.
3. Builds measurement sequence.
4. Initiates acquisition.
5. Stores acquired data and full copy of the Corrections and QWTB alg. setup to the measurement folder.
6. When requested by user, initiates processing of the acquired waveforms.
7. Signalizes “new result available” to the GUI process.
8. Repeats from (iv) until all acquisitions are done or user terminates the process by “STOP” button.

When “**GUI Process**” receives notification of the new result or user requires refresh of the results view, it will initiate refresh of the results view according to the current view setup by initiating another process “**Results Viewer Process**”. This process will search the measurement folder and will update the results view or initiates export of the measurement report. Note this process requires Matlab/GNU Octave, because the actual post-processing and formatting is done in Matlab/GNU Octave. The split into the processes means they can partially run in parallel, so when the digitizers are acquiring new waveforms, the “**Results Viewer Process**” can simultaneously perform the post-processing and displaying. The user can even plot graphs of the so far measured results during the measurements.

Finally, TWM contains “**Server control process**”, which allows to control some of the TWM functions and query status and data. The communication happens via Windows named pipe, so it can be controlled from any environment. The key point of this feature is the TWM can be controlled by another application that e.g. performs sequence of measurements. However, note the interface is in development stage and it is not part of the TracePQM project. Thus, it may not be fully developed before end of the project so it will be documented separately when it is ready to use.

## GOLPI

The communication between LabVIEW and Matlab/GNU Octave is ensured by the GOLPI interface [4]. The interface was designed for bidirectional runtime communication between LV and GNU Octave. The communication happens via the pipes which transfers commands and data between the two environments. User can also inspect the communication in console window. The pipes are based on the DLL library “lv\_process.dll” which is part of the project [4]. The “lv\_process.dll” can be used in any language such as CVI. However, it ensures just a low level text data exchange. Variables transfer between the LV and GNU Octave is done at LV level.

The project TracePQM also calls for a communication with Matlab which is far more popular among the potential users. Therefore, the GOLPI library for LV was modified so it also enables almost identic communication with Matlab via Matlab Script nodes. The nodes are hidden in the GOLPI so from outside there is no difference between use of GOLPI for Matlab and GNU Octave and there should be no difference apart from the performance, which may differ significantly. The only functional difference may be in some algorithms, where Matlab and GNU Octave implementation differs (see algorithms documentation).

## Control and data acquisition module

This module consists of two sub-modules: (i) Control (user interface GUI), (ii) Acquisition.

### Control module

Description to be done…

### Acquisition module

Acquisition module runs in a separate process (see Figure 0‑1).

Description to be done…

#### Modular driver design

The project objectives call for a modular driver concept. The key idea is the Acquisition module does not access the drivers of the particular instruments directly, because each digitizer requires completely different approach. Therefore the TWM tool would have to use different structure to work with different digitizers. So it was decided to insert a command translation layer in between the acquisition module and the drivers of physical instruments. This layer was called virtual digitizer. All HW specific function calls of each digitizer are translated to a universal format and merged into a few basic VI functions which are, for the acquisition module, identic for any digitizer no matter how different is the HW control implementation inside. The basic block diagram of the TWM in current version is shown in Figure 0‑2.

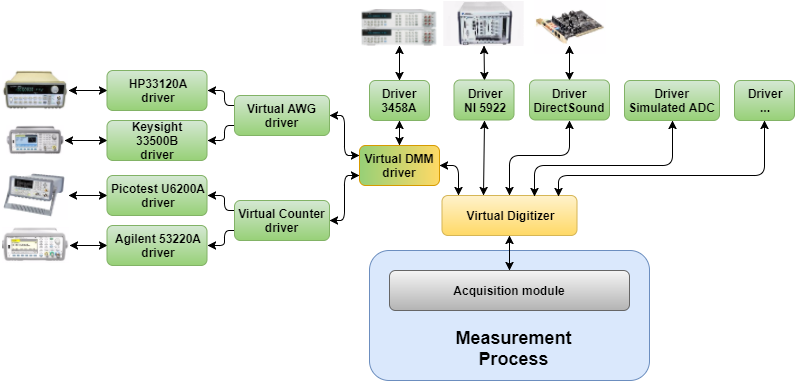


Figure 0‑2: Block diagram of TWM Virtual drivers.

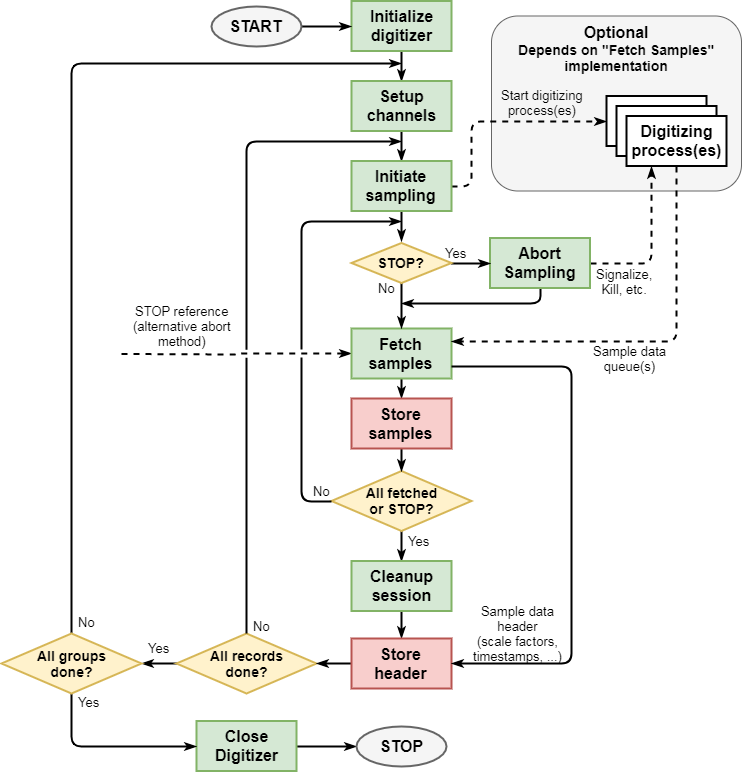


Figure 0‑3: Virtual digitizer driver structure and data flow. Green: virtual driver functions; Red: TWM acquisition module; White: Instrument specific part.

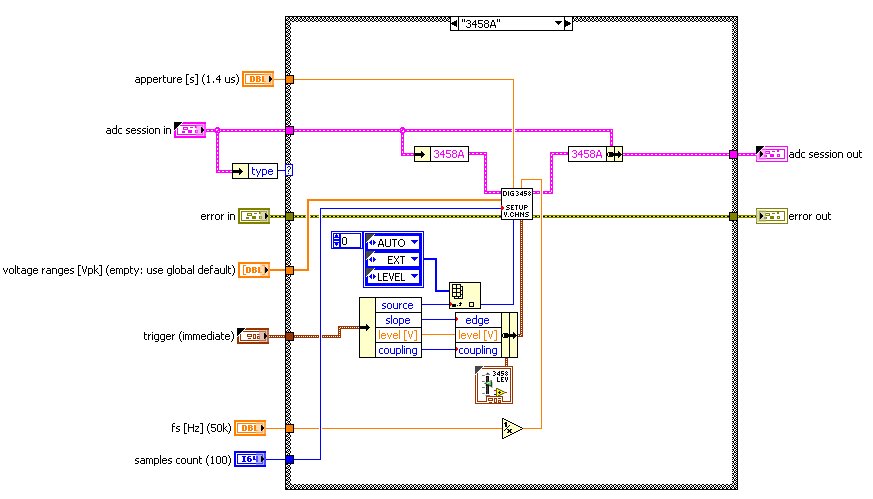
Detailed view on the driver structure and its usage in the TWM acquisition module is shown in Figure 0‑3. The virtual driver functions are shown in green colour. It was decided the driver should not directly write the sample data file, because whenever format of the data changes, each driver would have a different implementation. That is not effective and clean solution. The amount of data in the streaming mode can easily exceeds memory limit, which is just around 1 GB for the 32bit LabVIEW, so the driver cannot simply collect the sample data and then send them to TWM acquisition module at once for saving. Therefore, a rather complex solution capable of runtime storage of sample data based on the background digitizing process(es) was developed. Thanks to the acquisition in the separate process(es) the main fetch loop is non-blocking. Fetching and storing of the sample data runs continuously, so just a limited memory buffers are needed. The acquisition module can easily refresh sampling status and terminate it at any point even if the HW drivers does not allow that directly (e.g. by killing the process(es)). Furthermore, the execution priority of the digitizing process(es) was increased. This way the digitizing runs unaffected by the workload of rest of the application. That may be crucial for the time critical 3458A streaming mode and for high speed streaming from the NI 5922 cards. The throughput was tested and the limiting factor was HDD, which limited the write speed to some 120 Mbytes/s. However, as will be shown in the following chapters, the drivers for other, simpler digitizers do not need to use such a complex structure.

#### Virtual driver functions structure

Following chapters describes particular functions of the virtual digitizer, describes their inputs and outputs, behaviour and also explains where and when are they called by the TWM acquisition module.

To implement a new driver one must adapt and merge the low level instrument driver functions so they principally fit into the functions below (resp. the green coloured blocks in Figure 0‑3). Note there are also a few more functions that need to be implemented apart from the functions shown in the Figure 0‑3.

The virtual driver is a just a wrapper layer that translates the standardized inputs and outputs to the particular instrument drivers. See example for “Setup channels” for DMM 3458A driver:



Each of the following functions contains the case selectors with one item per digitizer “type”. Each function extracts the session related to the particular digitizer (“3458A” in the example) from the virtual digitizer “adc session”, it executes the function(s) of the instrument driver(s) and it stores the modified session “3458A” back to the virtual digitizer “adc session”. Other functions are made the same way. The only exception is the configuration panel for the digitizers which will be described separately.

There are just a few steps to be done to integrate new drivers. First, change the type definition of “type” in the “adc session”, i.e. add a new item to “type” Enum. The item names should be chosen clearly, such as “NI 9234”. Next, a session cluster (or class) of the new driver must added to the “adc session”. This object can contain absolutely anything. It depends on the driver. Finally, each of the case selectors in each of the virtual driver functions must be extended by the new page, e.g. “NI 9234” and the driver functions must be inserted. TWM will then automatically allow to use the new digitizer without any changes in the rest of the application.

##### ADC session

“ADC session” is a virtual digitizer cluster that has to contain all sub-sessions of the particular digitizers. It also contains several common items. Details on content of this cluster at the time of writing this document (may extend in future):

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| niScope | cluster | 5922 (niScope) driver session. |
| 3458A | cluster | Virtual DMM 3458A driver session. |
| Dsdll | cluster | DirectSound driver session. |
| simadc | cluster | Simulated ADC session. |
| \* | \* | New driver sessions… |
| Type | enum | Selected digitizer type {‘niScope’,’3548A’,’DirectSound’,’dummy’}. |
| modified? | bool | Flag set by TWM to “True” when HW setup was modified. |
| channel idn str | 1D array of string | Array of last queried identifier strings of particular channels of selected digitizer. One item per channel. |
| aux instr idn str | 1D array of string | Array of last queried identifiers of auxiliary HW related to the selected digitizer (e.g.: AWG, Counter, etc.). |

Adding a new digitizer means the session of the digitizer driver will be added and “type” enum will be redefined to contain unique identification name of a new digitizer. The rest must not be changed.

#### Virtual driver function reference manual

##### Initialize driver (optional)

Some digitizer drivers may need to perform some step to make them usable. This optional function is called once automatically on the TWM startup.



Function inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Enumerate devices (optional)

This optional function is called manually in the digitizer selection panel. It was added, because some of the drivers may require additional manual refresh of the installed HW configuration. It was used for the DirectSound drivers where it enumerates available input capture devices.

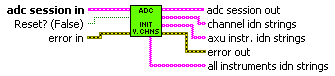


Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| capture devices ring ref | in | reference to ring | Reference to a ring control to be filled with enumerated devices. |

##### Initialize digitizer (required)

It is first function called by TWM before new measurement. Its purpose is to initialize and identify all HW components related to the digitizer. E.g.: for virtual digitizer based on the 3458A multimeters it is one or more 3548A units and optionally a pulse generator AWG or a counter. The function also sets the parameters which are not expected to change during the whole measurement session, such as mode of sampling (“DC V”, “DSDC”, …), coupling, etc. It always returns unique and clear identifiers of the channels and auxiliary HW.

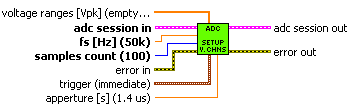


Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| reset? | in | bool | Force reset of the instruments? Similar to standard instrument driver template. |
| channel idn strings | out | 1D array of strings | Queried identifiers for each virtual channel. |
| aux intr. idn strings | out | 1D array of strings | Queried identifiers of additional HW related to the selected digitizer. |
| all instruments idn strings | out | 1D array of strings | All identifiers merged to one array. |

##### Setup channels (required)

This function is called once per group of measurements. I.e. it is not recalled before each repetition cycle so the driver must be prepared to perform several acquisition without recalling this. It will configure the virtual channels of and the virtual digitizer to the desired setup prior the acquisition. This function sets sampling rate, sample count, aperture, triggers, ranges, etc. The digitizer shall be ready to start acquisition after this function call.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| fs [Hz] | in | double | Desired sampling rate in [Hz]. |
| samples count | in | double | Desired samples count per channel. |
| trigger | in | cluster | Trigger setup cluster. |
| aperture [s] | in | double | Desired aperture of the ADC. Note this parameter will be ignored if digitizer does not support it. |

Cluster “trigger” contains following items:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| source | Enum | Trigger type {‘Immediate’ – start immediately; ‘External’ – from external trigger input; ‘Level’ – input channel level trigger}. Note the ‘Level’ is always related to the first channel. This may be eventually configured in driver specific configuration panel. |
| slope | Enum | Trigger slope sensitivity for “Level” and “External” triggers {“POS” or “NEG”}. |
| coupling | Enum | Coupling of the “Level” trigger {“DC” or “AC”}. Eventual other configurations must be handled by the driver itself and set from configuration panel. |
| level [V] | Double | Trigger level for “Level” mode in Volts. |

##### Initiate sampling (required)

This is when TWM is ready to digitize. This function should immediately initiate the sampling (arm the virtual digitizer) and return. The actual operation depends on the implementation of the “Fetch samples” function (see below).



Input and outputs:

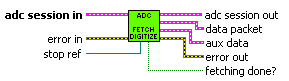
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Fetch samples (required)

The function is called in the loop to fetch the sample data and status. Purpose of this function is to obtain the acquired data from the virtual digitizer channels. There are three basic options for its implementation depending on the particular digitizer:

1. Blocking function that won’t return until all samples were acquired. This is suitable for smaller counts of samples, however the cost for this solution is TWM cannot query state of the sampling and the termination by STOP command is harder to implement (or impossible) as well as timeout. This is typical way the most of the instrument drivers are made.
2. Asynchronous function that just checks weather the sampling is done and eventually returns available samples block. When not done, it will just return status if possible. This way the sampling loop is not blocked and the TWM can update sampling status and easily terminate sampling by STOP command. This mode is however not possible for all digitizers as some of them do not allow asynchronous operation.
3. Complex implementation shown in the Figure 0‑3 where the “Initiate sampling” just starts the “digitizer process(es)” and the “Fetch samples” periodically checks, weather there are a new sample data available. If so, it returns block of samples that is stored to the file by “Store samples”. It is hard to implement, but it seems to be very useful for the 3458A streaming driver and for high speed PXI 5922 driver because the process(es) may be run with increased priority. This should prevent overflows for high speed digitizing or time critical digitizing (3548A).

Note this function always receives LV reference to a global Boolean variable “STOP”. This reference can be used as an alternative way to terminate the sampling (default is “Abort Digitizing Process”). Note the driver function must not change the values of “STOP”. It is just for reading.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | Cluster | Virtual digitizer session. |
| adc session out | out | Cluster | “adc session in” copy with eventual changes. |
| error in | in | Cluster | Error signal. |
| error out | out | Cluster | Error signal. |
| stop ref | in | Reference to bool | Reference to the global Boolean variable STOP. The variable becomes “True” when stop is requested. The function cannot modify the flag. |
| data packet | Out | Cluster | Cluster with block of sample data. |
| aux data | Out | Cluster | Cluster of additional data returned by the driver. |
| fetching done? | Out | Bool | This flag must be “True” when sampling is finished or it was terminated. |

The “data packet” is a cluster containing following:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| int32 | 2D array of int32 | 2D array of samples. One column per channel. |
| int16 | 2D array of int16 | 2D array of samples. One column per channel. |
| is int16? | Bool | Defines which of the “int16” or “int32” is valid. The other must be empty. |
| all done? | Bool | “True” when all samples were fetched. |
| sampling? | Bool | “True” when digitizing is in progress. |
| valid? | Bool | “True” means the other items are valid. Otherwise they are ignored by acquisition module. This may indicate the iteration of fetching was returned no data. |
| instr buffer [%] | Double | Indicates utilisation of the digitizer buffer. This is e.g. used for the 5922. May be “NaN” is not supported. |
| queue buffer [%] | Double | Utilisation of the data queue between digitizing process(es) and fetch function. May be “NaN” if not supported. |
| offset [smpl] | Int64 | Offset of the first sample in the block from start of the acquisition. Counting from zero. |
| count [smpl] | Int64 | Samples count in the sample array per channel. May be zero if no data fetched. |

The “aux data” content:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| T smpl | 1D array int64 | Indices of the samples to which the temperature readings are aligned. |
| T [deg C] | 2D array of doubles | 2D array of temperature readings during the acquisition. One column per channel, one row per item of “T smpl”. Note this is optional and the arrays may be empty. |
| Time stamp [s] | Double | Relative timestamps returned by the channels. These are relative time intervals in Seconds of the first sample of each channel related to some common event, e.g. reset of 5922. |
| Gain [V] | 1D array of doubles | Gain factors to get voltage from the integers in data packets. One per channel. |
| Offset [V] | 1D array of doubles | DC offset to add to real samples to get actual voltage. One per channel.  u(k) = gain\*y(k) + offset; u – voltage, y – integer sample |
| Increment [s] | Double | Sampling period [s]. |
| Valid? | Bool | “True” means the other items are valid. Otherwise they are ignored by acquisition module. Note the driver may return this cluster valid any time during the sampling. Whenever the flag is set, acquisition module remembers the data. So it does not matter if it returns at the start or the end of sampling. |

##### Cleanup session (required)

This function should terminate everything that may have left in the memory/system after the “Initiate sampling” function, e.g.: the processes, queues, shared memory, etc. This is called by TWM every time to cleanup after acquisition (even terminated or failed).



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Close digitizer (required)

This function is called after acquisitions are done even in case of error or termination. This function should put all affected instruments to some default safe state and close opened sessions to them. It is strongly recommended to put the instruments to such a state so they cannot be damaged. E.g.: not 50 Ω input, higher range, etc. Also it is good practice to turn all instruments programmatically to the local control as some of them may not even have “Local” button.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Reset to preset mode (True) | In | Bool | “True” means the function should reset the digitizer instruments to some safe default state before closing the handles. |

##### Abort Digitizing Process (recommended)

This function is called in the fetch loop when GUI signalizes STOP command. Implementation depends on the “Fetch samples” variant. For variant (i) it cannot be used as the function is blocking. For the other two variants, it may either send the signal to the digitizer if it supports such a function, or it can kill the digitizing process(es) (variant iii). Naturally “Fetch samples” must be able to recognize the digitizing process(es) were terminated and also signalize sampling done so acquisition module will leave the fetching loop.

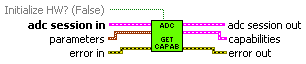


Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Send abort signal? | In | Bool | Sends abort only when “True”. Nothing happens when “False”. |

##### Get Digitizer Capabilities (required)

This function returns capabilities of the selected digitizer. It is called at various places of the TWM. It should NOT touch the HW by itself! All HW related information shall be obtained in the “Initialize digitizer” function and kept in the digitizer session even after “Close digitizer” is called! TWM decides by itself when to call “Initialize Digitizer”+”Close digitizer” to refresh the parameters so the time consuming querying is not performed when it is not needed. The capabilities are used to limit the GUI entries and disable the unsupported features. The driver should just query the information from the session and return the desired capabilities.



Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Initialize HW? | In | Bool | Forces new query of the instruments capabilities. Otherwise uses last queried capabilities from digitizer session (fast mode). |
| Parameters | In | Cluster | Cluster of some parameters that may be needed to obtain the capabilities. |
| Capabilities | Out | Cluster | Cluster of obtained capabilities. |

Cluster “parameters” contains following:

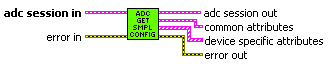
|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| fs [Hz] | Double | Current sampling rate in [Hz |
| aperture [s] | Double | Aperture time [s]. |

Cluster “capabilities” contains following:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| Max samples count | Int64 | Maximum number of samples to be acquired in one record. |
| Max fs [Hz]  Min fs [Hz] | Double | Maximum and minimum sampling rate [Hz]. |
| Ts step [s] | Double | Available sampling period step in which the rates can be set. |
| fs step [Hz] | Double | Available sampling rate step that can be set by digitizer. Note either “Ts” of “fs” step can be used. The other must be “NaN”. |
| Smpl rate step mode | Enum | Mode of sampling rate selection {‘const period’,’const frequency’}. Selects which of the “Ts” of “fs” is valid. |
| Max Ts [s]  Min Ts [s] | Double | Maximum and minimum sampling period [s]. |
| Aper min [s]  Aper max [s] | Double | Minimum and maximum apertures [s]. Or “NaN” if not supported. |
| Channels count | Double | Number of virtual channels available configured. |
| Allows streaming? | Bool | Set when the driver supports two modes: memory buffer and direct streaming. If it supports just one, it is ignored. |
| Streaming on? | Bool | Streaming mode configured. |
| Has level trig? | Bool | Driver/digitizer supports level triggering. |
| Has ext trig? | Bool | Driver/digitizer supports external input triggering. |
| Has aperture? | Bool | Driver/digitizer supports setting the appertures. |
| Has ranges? | Bool | Driver/digitizer can set multiple ranges. |
| Has temperature? | Bool | Driver/digitizer supports temperature measurement. |
| Has temperature log? | Bool | Driver/digitizer supports temperature logging during acquisition. |
| Has selfcal? | Bool | Driver/digitizer supports self-calibration routine. |

##### Get Current Setup (required)

Similar to the “Get Digitizer Capabilities”. It should not touch the HW. It should return last used configuration from the digitizer session. This function returns two groups of parameters. First, the standard ones, e.g.: sampling rate, samples count, trigger, etc. Next, the specific for given digitizer.



Function inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | Out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | Out | cluster | Error signal. |
| Common attributes | Out | Cluster | Standard attributes/parameters of the digitizer. |
| Device specific attributes | Out | 1D array of clusters | 1D array of clusters containing:  “name” – attribute name string  “value” – 1D array of test string with formatted value  “is constant per group” – when trues, the attribute is stored just once for measurement group. Otherwise it is stored for each record.  Note the value may be numeric. In that case format the number to decimal, floating or exponential format with decimal dot “.”. These attributes are not used by TWM anywhere, they are just store as additional items to the measurement session header. They will appear there as “name:: value” or as:  #startmatrix:: name  value(1); value(2);…  #endmatrix:: name  Note the “aperture” value is one of these attributes. |

Cluster “common attributes” contains following items:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| Count [smpl] | Int64 | Configured number of samples to acquire per channel. |
| Channels count | Int32 | Number of configured channels in the virtual digitizer. |
| fs [Hz] | Double | Configured sampling rate in [Hz]. |
| Ranges [V] | 1D array of doubles | Array of set range values as defined by the particular drivers. One value per virtual channel of the digitizer. |
| Trigger | Cluster | Trigger setup, see above. |
| Ext freq. locked? | Bool | Status of PLL lock if supported. |
| Streaming on? | Bool | Set when streaming is enabled. |
| Is int16? | Bool | Set when data is/will be in int16 format for the configured sampling setup. |
| Bitres | Int32 | Actual bit resolution (how many bits are utilised in the integer). |

##### GUI Get Info (recommended)

This function takes digitizer session and returns a brief description of the digitizer which is displayed in the digitizer panel. It must not touch the HW. It may contain e.g. trigger connection notes (3458A mode). It is called in the digitizer configuration panel.



Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| Configuration description | Out | String | String with brief description of the current configuration of the digitizer. |

##### Selfcal Virtual Channels (optional)

This function should initiate self-calibration of the digitizer HW components if such function is supported. It is synchronous operation. TWM is blocked during its execution.

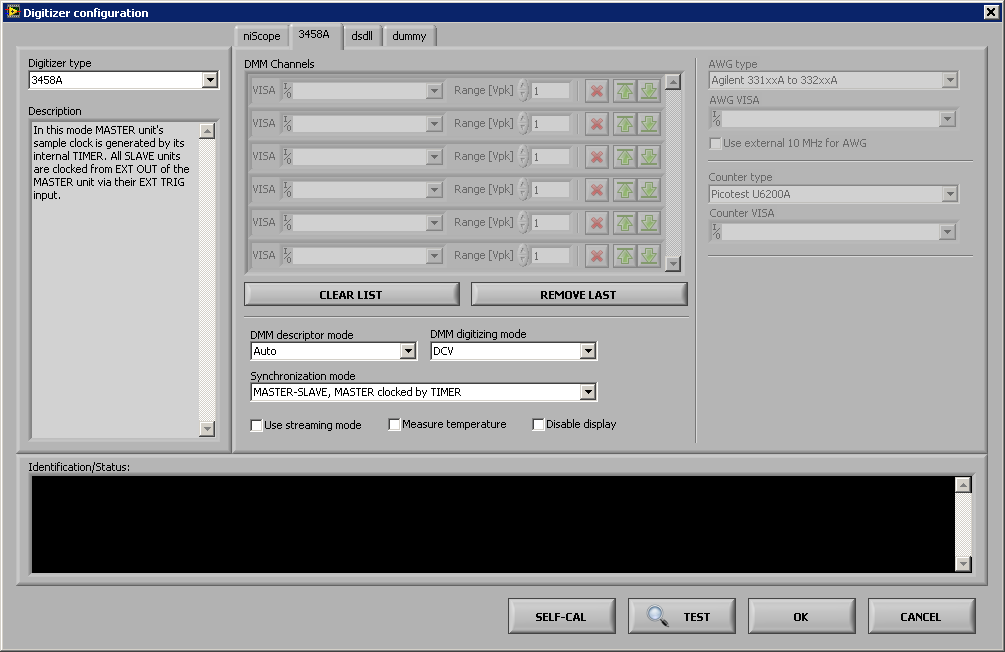


Inputs and outputs:

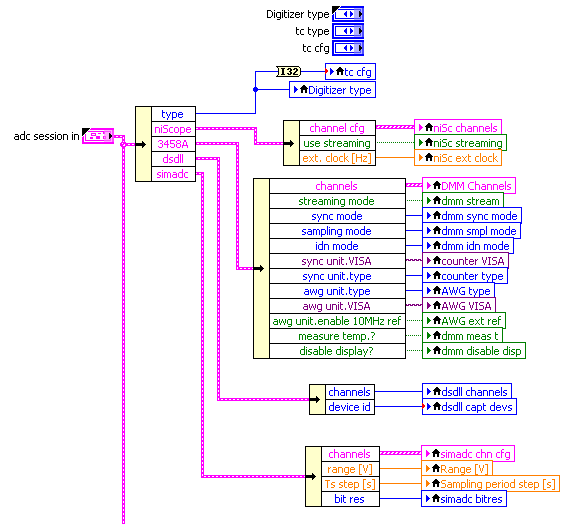
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | Out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | Out | cluster | Error signal. |

##### Digitizer configuration panel (required)

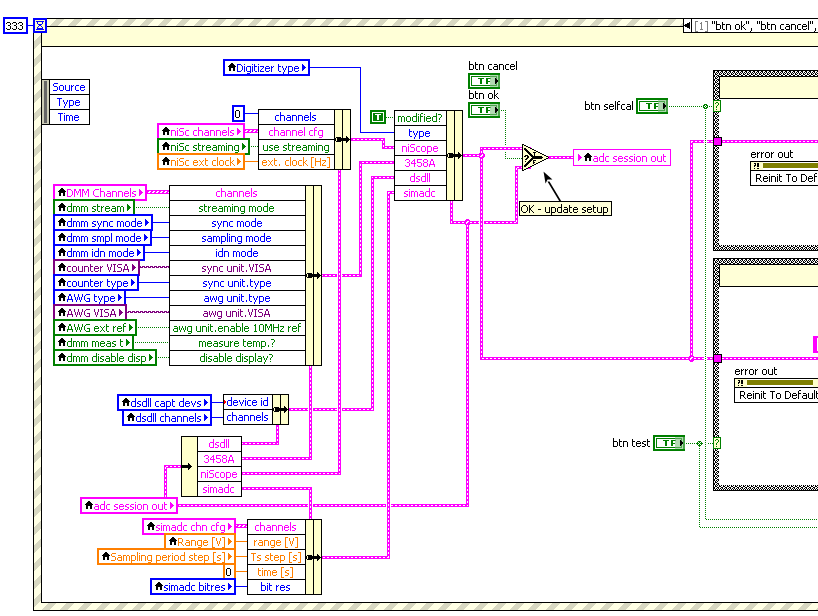
The digitizers in the TWM must be configured before they can be used. The configuration is done in a panel “Digitizer configuration”. The panel contains page control with one page per digitizer:



A new page must be added to implement a new digitizer. The order of the pages must match the order in the “type” Enum in the “adc session”. The new page can contain any configuration needed for the new digitizer. On initialization the current settings from the digitizer sessions must be set to the panel items related to the particular digitizers:



Accordingly before exiting the panel, the new settings must be stored back to the “adc session”:



The panel itself is based on the Event structure. Most of the events are common for all digitizers and thus doesn’t need to be modified. However, each digitizer may have some special requirements which must be placed in the new event. Example for DirectSound driver:

