TELEDYNE SP DEVICES
Everywhereyoulook™

Application Note:

Time stamp

Table of Contents

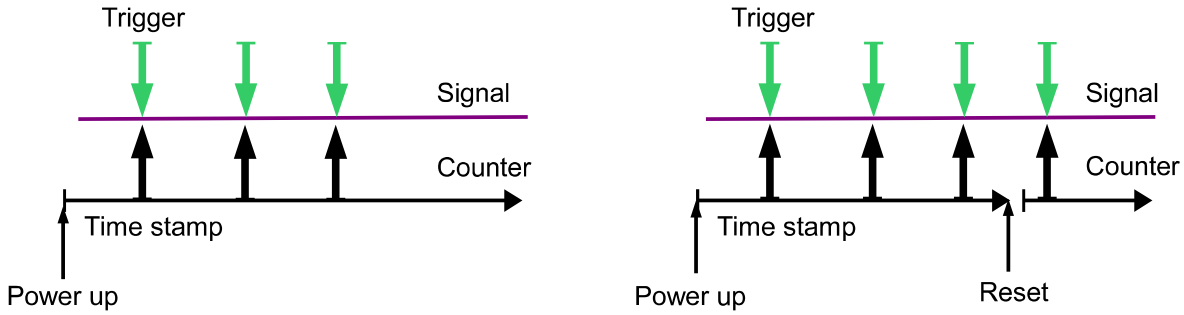
Introduction....2
General Technical Description....2
Application modes....3
  Sync Off....3
  Sync On 4
    Single restart:....4
    Repeated restart:....4
  Counter restart....5
  Time stamp vector....5
    To access vector (MATLAB)....5
Equations 6
  Translate time stamp to real time....6
An example: To calculate delta sync time....7
C/C++ Example Code....8
  Open Files....8
  Set Trigger Mode 8
  Set MultiRecord Mode....8
  Set TrigTime Mode....8
  Reset Trig Timer....8
  Get Trig Time....9
  Sync Off....9
  Sync On....9
  Disarm trigger....10
  Multirecord close....10
Use the result....10
    Collect data to PC....10
    Get delta sync time....10
MATLAB Example....12
To get the real trigger time....14
  Plot from MATLAB example above....16
C Example....18

Introduction

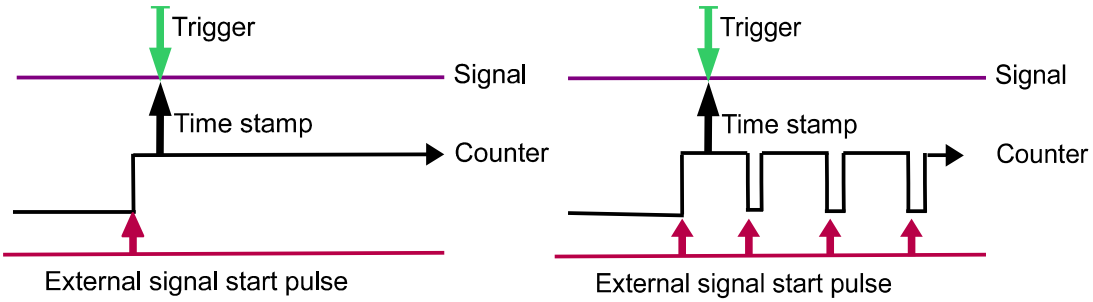
The ADQ has a 64 bit running time counter that enables the user to establish exactly when a certain set of data was captured, a feature which can be used eg. when calculating the velocity of particles in certain reaction sequence measurements.

This counter enables a time stamp for each sampling event, which means that at each trigger event the counter value is read and stored together with the recorded data. The time counter starts at power up, and can be reset by API SW command or by a pulse into GPIO pin2.

The counter can also be synchronized with an external pulse on GPIOpin2, and can then operate in two modes; single start signal or repeated restart signal.



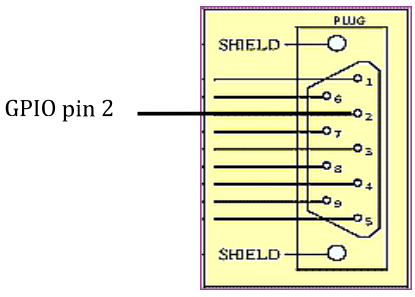
Ill. 1: Time Stamp



Ill. 2: External sync pulse, Single Start resp. External sync pulse, Reperated Restart

General Technical Description

Time Stamp Mode can be used with all the trigger modes supported by the digitizer. The counter starts, by default, at power up and it may directly be used for relative time measurement.
The external sync pulse-start/reset pulse is connected to GPIO pin 2. The time stamp data is stored according to which sync mode is used; if sync mode is off, the whole 64 bits of the counter vector is used for the time stamp counter. If sync mode is on, time stamp data consists of two parts; time stamp data and sync counter data; and the vector is divided into two sections: Sync counter in the higher 22 bits (42 - 63)



Ill. 3: GPIO-pin 2

and timer counter in the lower section (0-41) (see section TimeStampVector).

Application modes

Time Stamp can work in two modes: No external sync pulse ***or*** External sync pulse connected to GPIO pin2.

- Sync pulse off. (Sync Off)
  No external sync pulse. The counter starts as by default at power up and it may be reset by software reset command (API command).
- Sync pulse on. (Sync On)
  The counter is synchronized by an external pulse. (on GPIO pin2)
  This application mode operates in two different modes: time counter single restart and time counter repeated restart, se section below.

The application mode is set by command `SetTrigTimeMode`:

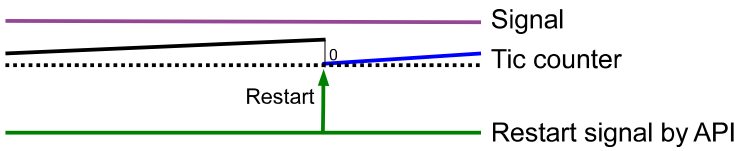
```
int ADQxxx_SetTrigTimeMode( void* adq_cu_ptr, int adqxxx_num, int TrigTimeMode )
```

```
ADQ_SetTrigTimeMode(adq_cu,1,0);
TrigTimeMode 0    = continious count, sync off
TrigTimeMode 1    = sync on. Activate sync mode, count sync
      pulses and  reset counter.
```

The output from the time stamp depends on which mode the time stamp runs in, se SyncOff and SyncOn section respectively.

Sync Off

In this mode the full 64 bits of the time stamp vector are used to represent the number of tics that the counter has achieved.
If the counter needs to be restarted, this is done by an API command.



Ill.4: Sync Off Restart by API command

The output from `GetTrigTime`

```
int ADQxxx_GetTrigTime( void* adq_cu_ptr, int adqxxx_num )
```

```
ADQ_GetTrigTime(adq_cu,1);
```

is in *SyncOff* mode represented by: $syncs \times 2^{42} + cycles \times 2^{2} + start_{val} + trig_{val}$

```
Cycles = ADQ_GetTrigTimeCycles(adq_cu,1)
Syncs = ADQ_GetTrigTimeSyncs(adq_cu,1)
start = ADQ_GetTrigTimeStart(adq_cu,1)
```

Sync On

With external start pulse (Sync On), there are two operational modes, single restart and repeated restart.

In sync on mode the lower 42 bits are used for the tic counter, while the upper 22 bits are used for the sync counter.

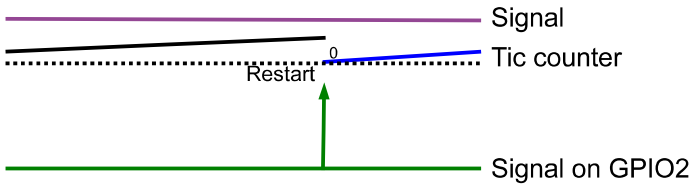
The output from `GetTrigTime`

```
int ADQxxx_GetTrigTime( void* adq_cu_ptr, int adqxxx_num )

ADQ_GetTrigTime(adq_cu,1);
```

is in SyncOn mode represented by: $cycles \times 2^2 + start_{val} + trig_{val}$

Single restart:

The restart pulse is connected to GPIO pin 2, and at a signal at GPIO2 the counter is reset.

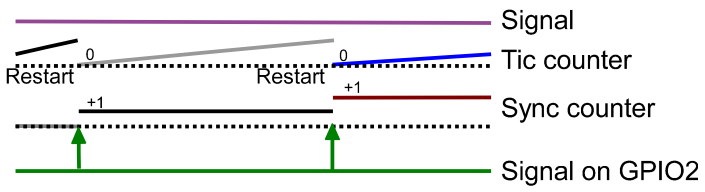


Ill. 5: Tic counter, External signal, Single restart.

Whether the counter is started immediately or if it waits for a start pulse is determined by the setting of `ResetTrigTimer`, se section Counter Restart.

Repeated restart:

The GPIO pin2 is used as start pulse input, to generate sync pulses. At each sync pulse the tic-counter is reset and the sync counter is incremented.



Ill. 6: Tic counter and Sync counter, Repeated restart, Sync On

Whether the counter is started immediately or waits for a start pulse is determined by the setting of `ResetTrigTimer`, se section Counter Restart.

Counter restart

The counter can be restarted in two ways; either by software, i.e. API-command, or by sending a pulse into the GPIO Pin2, sync input. The reset-mode is stated by command `ResetTrigTimer`.

If `TrigTimeRestart` is set to 0, the timer is reset at command, and then waits for a start pulse to start over again.
If `TrigTimeRestart` is set to 1, the timer is reset at command and then immediately restarts, without the need for a start-pulse.

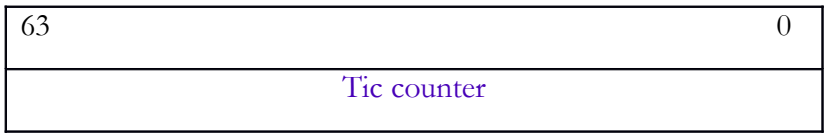
```
int ADQxxxResetTrigTimer( void* adq_cu_ptr, int
adqxxx_num, int TrigTimeRestart)
```

```
(ADQ_ResetTrigTimer(adq_cu,1,1));

TrigTimeRestart 0 = Reset, Waiting for start pulse
TrigTimeRestart 1 = Reset, Immediate restart of timer
```

Time stamp vector

The state of the sync mode activation in `SetTrigTimeMode` affects the properties of the **time stamp vector**:

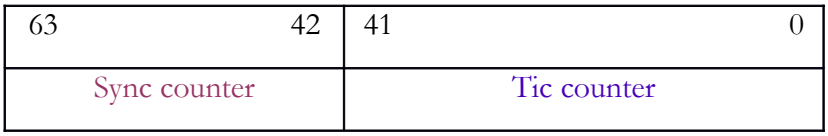
If mode is *Sync Off*:



Ill. 7: Time stamp vector when sync is not active

If sync mode is not activated, Sync OFF, all 64 bits of the time stamp vector are dedicated to the tic counter. At each trigger event, the state of the tic counter is read and stored together with the signal data.

If mode is *Sync On*:



Ill. 8: Time stamp vector when sync is active

If sync mode is activated, Sync On, 22 bit are allocated for the sync counter, 42 bit to the tic counter, which means that data stored together with signal data consists of both sync data (how many times the counter has been reset) and the actual time stamp from the tic counter.

Equations on how to translate the time stamp to real time is provided below, in section Equations.

To access vector (MATLAB)

To access beginning of lower part (bit 0-41) of vector (Tic counter) use mask

```
m_TrigVector[0] = (0x00000000f & m_MultiRecordHeader[2]);
```

Likewize, to access beginning of higher part (bit 42 - 63) of vector (Sync counter):

```
m_TrigVector[1] = (0x000f00000 & m_MultiRecordHeader[2])>>16;
```

Equations

Translate time stamp to real time

The time stamp is really just a counter. To really see what time has elapsed, this has to be converted to real time, using the equations below.

$$t = real\ time \qquad n_{tic} = tic\ counter \qquad f_s = ADQ\ sampling\ frequency$$
$$f_{sync} = sync\ pulse\ sampling\ frequency \qquad n_{sync} = sync\ counter$$

Sync off	Sync on
$t = \frac{n_{tic}}{f_s \times 2}$	$t = \frac{n_{sync}}{f_{sync}} \times \frac{n_{tic}}{f_s \times 2}$

C:

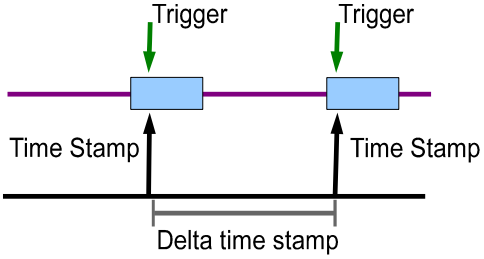
n_{tic} (ADQ_GetTrigTime(adq_cu,1))
n_{sync} (ADQ_GetTrigTimeSyncs(adq_cu,1))

MATLAB:

n_{tic} interface_ADQ('gettrigtime')
n_{sync} interface_ADQ('gettrigtimesyncs')

An example: To calculate delta sync time

To find the time that has elapsed between a pair of trigger events, maybe to find if samples has been missed when measuring on a periodic signal, the time stamp delta can be calculated.



Ill. 9: Delta sync time

Delta value is $\frac{(c_{i+1} - c_i)}{2}$ where c is the time-stamp data at a given sample.

C/C++ Example Code

Open Files

```
// Files for output storage
FILE* outfileA;
FILE* outfileB;
outfileA = fopen("data_A.out", "w");
outfileB = fopen("data_B.out", "w");
```

Open files to store output data

Set Trigger Mode

```
//Set trigger mode:
(ADQ_SetTriggerMode(adq_cu, 1, trig_mode));


(ADQ_SetLvlTrigLevel(adq_cu, 1, trig_level));


(ADQ_SetLvlTrigFlank(adq_cu, 1, trig_flank));


(ADQ_SetLvlTrigChannel(adq_cu, 1, trig_channel));
```

Set trigger mode
trigger mode:
1 = SW mode, 2 = external mode, 3 = level mode

trigger level: on ADQ214 -8192 <= level <= 8191

trigger flank : 1 = rising, 0 = falling

trigger channel: 1 = channel A, 2 = channel B, 3 = any channel

Set MultiRecord Mode

```
//Set number of records:
printf("\nChoose number of records:\n");
scanf("%d", &number_of_records);

//Set number of samples
printf("\nChoose number of samples per record:\n");
scanf("%d", &samples_per_record);

//Setup multi records
(ADQ_MultiRecordSetup(adq_cu, 1, number_of_records,
samples_per_record));

(ADQ_DisarmTrigger(adq_cu,1));
(ADQ_ArmTrigger(adq_cu,1));
```

Set multi record mode
number of records
number of samples
setup multi record
(For more about MultiRecord, see Application Note Multi Record mode)

Set TrigTime Mode

```
(ADQ_SetTrigTimeMode(adq_cu,1,0));//continious count

(ADQ_SetTrigTimeMode(adq_cu,1,1));//activate sync mode
```

Sync off : 0 Continious count

Sync on: 1 Activates sync mode, count the sync pulses, reset counter.

Reset Trig Timer

```
(ADQ_ResetTrigTimer(adq_cu, 1, 0));

(ADQ_ResetTrigTimer(adq_cu, 1, 1));
```

Reset trig timer to 0.
0 = reset to 0, waiting for start pulse to restart
1 = reset to 0, immediate restart

Get Trig Time

```
ADQ_GetTrigTime(adq_cu, 1);

ADQ_GetTrigTimeSyncs(adq_cu, 1);

ADQ_GetTrigTimeCycles(adq_cu, 1);
```

Returns the time stamp counter value, depending on TrigTimeMode.
SYNC_ON=cycles*2^2+start_val+trig_val

SYNC_OFF=syncs*2^42+cycles*2^2+start_val +trig_val

Syncs return sync counter value
Cycles return cycle counter value

Sync Off

```
synctrig = ADQ_ReadGPIO(adq_cu,1);

ADQ_SetTrigTimeMode(adq_cu,1,0));//continious count
                                 //Sync OFF
ADQ_DisarmTrigger(adq_cu,1));
ADQ_ArmTrigger(adq_cu,1));
ADQ_GetTrigTime(adq_cu,1));

        do{
                (ADQ_GetTrigTime(adq_cu,1));
                od_timest.push_back(ADQ_GetTrigTime(adq_cu,1));
        }while (!synctrig);

        do{
                if (restart_mode == 1)
                        {
                                (ADQ_ResetTrigTimer(adq_cu,1,1));
                        }
                else if (restart_mode == 0)
                        {
                                (ADQ_ResetTrigTimer(adq_cu,1,0));
                        }
        }while (synctrig);
```

ADQ_SetTrigTimeMode(adq_cu,1,0)
0 = continious count, sync off

Sync On

```
synctrig = ADQ_ReadGPIO(adq_cu,1);


(ADQ_SetTrigTimeMode(adq_cu,1,1));//activate syncmode
                                  // Sync ON
(ADQ_DisarmTrigger(adq_cu,1));
(ADQ_ArmTrigger(adq_cu,1));

do
        {
                if (restart_mode == 1)
                {
                        (ADQ_GetTrigTime(adq_cu,1));

                        (ADQ_GetTrigTimeSyncs(adq_cu,1));

                        (ADQ_ResetTrigTimer(adq_cu,1,1));
                }

                else if (restart_mode == 0)
                {
                        (ADQ_GetTrigTime(adq_cu,1));
```

ADQ_SetTrigTimeMode(adq_cu,1,0)
1 = sync mode

```
                                (ADQ_GetTrigTimeSyncs(adq_cu,1));

                                (ADQ_ResetTrigTimer(adq_cu,1,0));
                        }
                }while(synctrig);
```

Disarm trigger

```
// Only disarm trigger after data is collected
(ADQ_DisarmTrigger(adq_cu,1));
```

When data collection is done, Trigger must once again be disarmed.

Multirecord close

```
// Return ADQ to single-trig mode
(ADQ_MultiRecordClose(adq_cu,1));
```

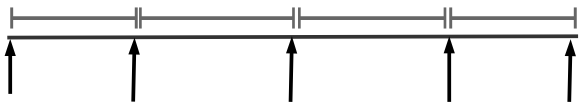
Return ADQ to default state

Use the result

Collect data to PC

```
// Get pointer to non-streamed unpacked data
        int* data_a_ptr_addr0 = ADQ_GetPtrDataChA(adq_cu,1);
        int* data_b_ptr_addr0 = ADQ_GetPtrDataChB(adq_cu,1);

        for (unsigned int i=0; i<n_records_collect; i++)
        {
        unsigned int samples_to_collect = samples_per_record;
        while (samples_to_collect > 0)
        {
        int collect_result = ADQ_CollectRecord(adq_cu,1, i);
        unsigned int samples_in_buffer = MIN(ADQ_GetSamplesPerPage(adq_cu,1),
samples_to_collect);

        if (collect_result)
        {
                for (unsigned int j=0; j<samples_in_buffer; j++)
                {
                        int dataa = *(data_a_ptr_addr0 + j);
                        int datab = *(data_b_ptr_addr0 + j);
                        fprintf(outfileA, "%d\n", dataa);
                        fprintf(outfileB, "%d\n", datab);
                }
                samples_to_collect -= samples_in_buffer;
        }

}
```

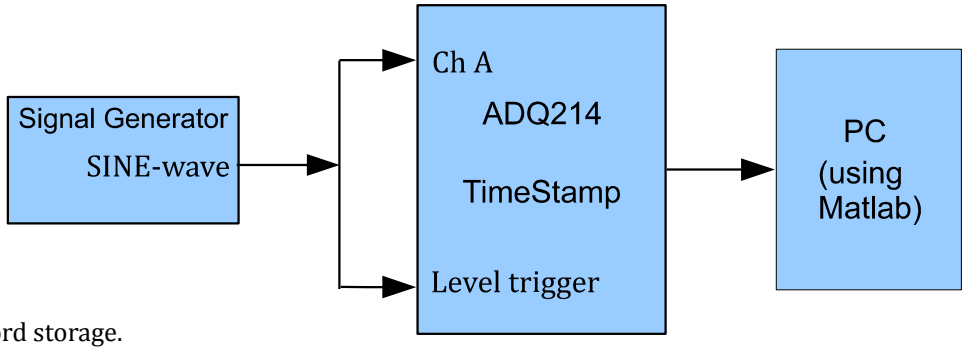
Get delta sync time



Ill. 9 delta SyncTime

```
vector<double>::const_iterator cii;
vector<double>::const_iterator cii_cal;
for(cii=od_timest.begin(); cii!=od_timest.end(); cii++)
{
	for(cii_cal=od_timest_cal.begin(); cii_cal!=od_timest_cal.end(); cii_cal++)
		{
			od_timest_cal.push_back((*cii+1 - *cii)/2);
		}
}
```

MATLAB Example



Multi record storage.
Level trigger.
Sync off.
ADQ 214.

```
% Time stamp
%
%
param = [];


selecti = 0;
clear data


    selecti = selecti+1;

 % Settings
    LevelTrigKeywordList = [{'CH_A'} {'CH_B'} {'CH_C'} {'CH_D'}];

    LvlTrigEdge = 'RISING_EDGE'; %RISING_EDGE or FALLING_EDGE
    LvlTrigCh = LevelTrigKeywordList{chi}; %NO_CH, CH_A, CH_B or ANY_CH
    SLvlTrigLevel=Settings.LvlTrigLevel*adqparam.rawdatascale;
    TriggerMode = 'LEVEL_TRIGGER_MODE'; %USB_TRIGGER_MODE,
EXTERNAL_TRIGGER_MODE or LEVEL_TRIGGER_MODE
    SampleWidth = adqparam.numberofbits;
    PreTrigSamples = 128;

 % Set up ADQ device
    interface_ADQ('settriglevelresetvalue',100);
    interface_ADQ('set_trigger_edge',1);

    %mex_ADQ([95 100]);
    %mex_ADQ([1 0]);

 % Setup multitrig, two records 1024 bytes
    nofrecords = 2;
    nofsamples = 1024;
    interface_ADQ('set_trigtimemode',0); % SYNC_OFF
    interface_ADQ('reset_trigtimer',1); % Restart timer on 0.
    interface_ADQ('multi_record_setup', [nofrecords
recordsize]);

 % Disarm and arm
    interface_ADQ('disarmtrigger');
    interface_ADQ('armtrigger');

    % Init variables
    triggered = 0;
    triggerWatchdog = 0;

    % Check for trigger
    while (triggered == 0 && triggerWatchdog<11 ) % 1s time out
```

Set Level Trigger mode

Set trigger reset value:100

Set trigger edge: rising

Set timer mode: sync off

Reset timer for immediate restart

Set multi record storage mode

```
      pause(0.1);
      triggerWatchdog = triggerWatchdog+1;
      [a,b,c,d] = interface_ADQ('getacquiredall'); % Check if all records triggered
      triggered = c(1);
    end

    if triggered == 0
      ResultKeyWord = 'FAILED!! not triggered';
    else
      ResultKeyWord = 'All triggered. Start analysis. ';
    end
    for fi = 1:length(SystemParam.fidlist)
      fprintf(SystemParam.fidlist(fi),'%s %s.\n', ResultKeyWord,  adqparam.ChannelNames{chi});
    end

    if triggered == 1
      for trigindex = 1:nofrecords
        %[data{1}, data{2}, status, validation] = interface_ADQ('collect_record', trigindex-1);
        [data, status, validation] = ADQ_PT_GetRecords(SystemParam,Spec,adqparam,trigindex-1);
        %if FoundRisingEdge == 0
        %  Settings.PreTrigSamples
        %end
        [a, b, status, validation]=interface_ADQ('gettrigtime');
        figure(fh);
        TrigTime(trigindex)= double(status(1));
        subplot(SystemParam.numberOfSelectedChannels,1,selecti)
        plot(ADQ_PT_ScaleData(adqparam,data{chi}(1:plotlength)),'color',rand(1,3)); hold on
        if data{chi}(FallingEdgeLength)<data{chi}(10)
          Valid(chi)=false;
          for fi = 1:length(SystemParam.fidlist)
            fprintf(SystemParam.fidlist(fi),'FAILED!! %s Found falling edge.\n',
adqparam.ChannelNames{chi});
          end
        end
        if trigindex == 1; figure(116); clf; plot(ADQ_PT_ScaleData(adqparam,data{chi}),'-.b');
title('ADQ_PT_TimeStamp 1st record'); end
      end
      title([adqparam.ChannelNames{chi}]);
      drawnow

          dTrigTime = diff(TrigTime)/2; % Unit [periods].  2 samples per period in TrigTime ADQ214.


      %dTrigTimeSigma = sqrt(var(dTrigTime))
      TrigFreqEstimate = mean(adqparam.samplingFreqInt(1)./dTrigTime);
      TrigFreqEstimateSigmaRel = sqrt(var(adqparam.samplingFreqInt(1)./dTrigTime))/TrigFreqEstimate;
      if 0.95>TrigFreqEstimate/timeTestFrequency || 1.05<TrigFreqEstimate/timeTestFrequency
        Valid(chi)=false;
        ResultKeyWord = 'FAILED!!!';
      else
        ResultKeyWord = 'PASSED.  ';
      end

      for fi = 1:length(SystemParam.fidlist)
        fprintf(SystemParam.fidlist(fi),'%s %s. %s Estimated freqeuncy %5.1f kHz. Relative sigma
%7.6f. Expected %5.1f kHz.\n', ...

testnumber,adqparam.ChannelNames{chi},ResultKeyWord,TrigFreqEstimate/1e3,TrigFreqEstimateSigmaRel,ti
meTestFrequency/1e3);
      end

      % Write to XLS file
      xlsinfo = {Valid(chi)};
      xlscolindex = find(strcmp(SystemParam.xlsColumnDefinition,{testnumber}));
      TestResult.xlsResultMatrix(chi,xlscolindex:xlscolindex-1+length(xlsinfo)) = xlsinfo;

      figure(26); clf;
      subplot(3,1,1)
      plot(TrigTime)
      title('Trig Time')
      subplot(3,1,2)
      plot(dTrigTime)
      title('diff Trig Time')
```

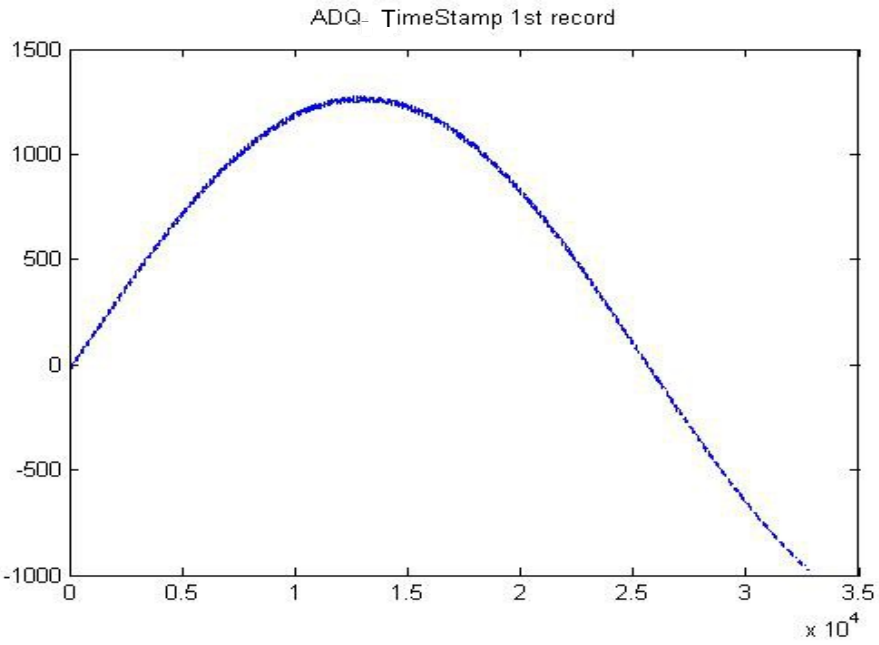
Get time stamp data

```
      subplot(3,1,3)
      plot(dTrigTime-mean(dTrigTime))
      title('Trig Time deviation')

    end

  end
end
```

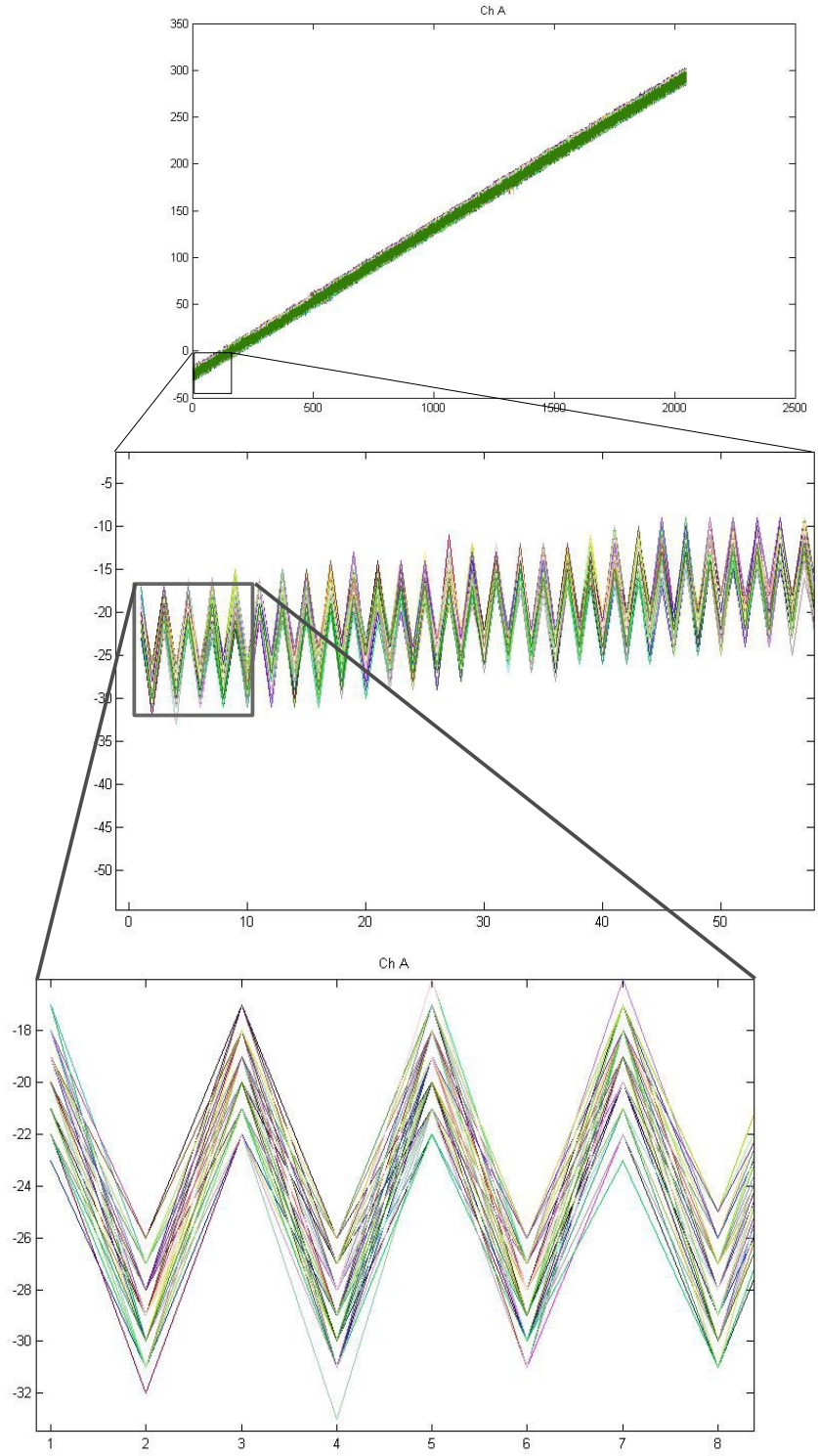
To get the real trigger time

```
% Collect data from ADQ
  [data_A, data_B]= interface_ADQ('collect_record', loopi-1);

% Read trigger information from multi record header
  [tmpa tmpb header tmpd]=interface_ADQ('get_multirecordheader');
  trig_information=header(3);

%  CALCULATE TRIGGER TIME OFFSET

% Bit 15 and 16 in the register. MATLAB index starts from 1.
% The trigger offset is in the form [0 1 2 3].
% This is translated to delay, which is a fraction of a sampling period [0.0 0.25 0.5 0.75].
trig_information = dec2bin(trig_information,32);  % Convert register  to vector of bits
trigger_offset = 0.25*(bin2dec(trig_information(15:16)));    % Unit [samples]
disp(['---ADQ214_accurate_trigger---Trigger offset ' num2str(trigger_offset) ' samples.'])
%trigger_offset = 0.25*(bin2dec(trig_information(11:12)))    % Unit [samples]
```

Plot from MATLAB example above



Ill. 10: ADQ Time Stamp, 5.2kHz input signal 4.1 dBm



Ill. 11: Plot from example above. Channel A

C Example

Multi record storage.
SW-level trigger
Sync off.
Find delta-trig time.

```
// File: example_ADQ214_timestamp.cpp
// Description: An example of how to use the ADQ-API (ADQ214)) in multi record and time stamp mode.
// Will connect to a single ADQ device and collect a batch of data into
// "data_A.out" and "data_B.out" (dual channel boards).
// Will put a time stamp on each sample.
// ADQAPI.lib should be linked to this project and ADQAPI.dll should be in the same directory
// as the resulting executable.

#define _CRT_SECURE_NO_WARNINGS

#include "ADQAPI.h"
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

void adq_time_stamp(void *adq_cu);

        // Special define
#define MIN(a,b) ((a) > (b) ? (b) : (a))
#define MAX(a,b) ((a) > (b) ? (a) : (b))

void adq(void *adq_cu)
{
        int* revision = ADQ_GetRevision(adq_cu,1);
        printf("\nConnected to ADQ #1\n\n");

        adq_time_stamp(adq_cu);
}

void adq_time_stamp(void *adq_cu)
{
        //Setup ADQ
        int trig_mode = 1; // SW trig mode
        int trig_level;
        int trig_flank = 1; // Rising
        int trig_channel = 3; // Any channel
        int clock_source = 0; // Internal clock
        int pll_divider;
        unsigned int number_of_records = 2; // Two records
        unsigned int samples_per_record = 1024; // 1024 samples
        unsigned int n_records_collect;
        int counter_mode;
        int time_stamp_mode;
        int trig_time_mode = 0; // Sync off
        int sync_mode;
        int synctrig;
        vector<double> od_timest;
        vector<double> od_timest_cal;

        // Files for output storage
        FILE* outfileA;
        FILE* outfileB;
        outfileA = fopen("data_A.out", "w");
        outfileB = fopen("data_B.out", "w");

        // Set trigger mode
        (ADQ_SetTriggerMode(adq_cu,1, 1)); //1 = SW trigger mode

        //Set clock source
```

Set up ADQ to SWtrigger mode, Rising flank, trig on any channel, internal clock, write to two records, 1024 samples and sync off mode.

Set files for output storage

SW trigger mode

```
        (ADQ_SetClockSource(adq_cu,1, 0)); //0 = Internal clock source

        printf("\nChoose PLL frequency divider.\n 2 <= divider <=
20, f_clk = 800MHz/divider\n"); //Sampling frequency
                scanf("%d", &pll_divider);
        (ADQ_SetPllFreqDivider(adq_cu,1, pll_divider));

        //Reset trig timer
        (ADQ_ResetTrigTimer(adq_cu,1,1)); //1 = Immediate restart


        //Setup multi records
        (ADQ_MultiRecordSetup(adq_cu, 1, number_of_records, samples_per_record));

        //printf("\nPlease trig your device to collect data.\n");

                (ADQ_DisarmTrigger(adq_cu,1));
                (ADQ_ArmTrigger(adq_cu,1));

                int trigged;
                do
                {
                        printf("\ntrigged.\n");
                        trigged = ADQ_GetTriggedAll(adq_cu,1);
                }while (trigged == 0);

        printf("\nDevice trigged\n");*/


        //sync mode set
        (ADQ_ReadGPIO(adq_cu,1)); // 1 = sync off
        synctrig = ADQ_ReadGPIO(adq_cu,1);

        (ADQ_SetTrigTimeMode(adq_cu,1,0));//0 = continious count
        (ADQ_DisarmTrigger(adq_cu,1));
        (ADQ_ArmTrigger(adq_cu,1));
        (ADQ_GetTrigTime(adq_cu,1));

                do{
                        (ADQ_GetTrigTime(adq_cu,1));
                        od_timest.push_back(ADQ_GetTrigTime(adq_cu,1));
                        }while (!synctrig);

                if (synctrig)
                {
                        (ADQ_ResetTrigTimer(adq_cu,1,1));
                }

        printf("Choose how many records to collect.\n 1 <= records <= %d, 0 == All in ADQ buffer.\n",
number_of_records);
        scanf("%d", &n_records_collect);

        if (n_records_collect == 0)
                n_records_collect = number_of_records; // Collect ALL records

        // Get pointer to non-streamed unpacked data
        int* data_a_ptr_addr0 = ADQ_GetPtrDataChA(adq_cu,1);
        int* data_b_ptr_addr0 = ADQ_GetPtrDataChB(adq_cu,1);

        // get trigger time delta
        printf("Collecting data, please wait...\n");
        for (unsigned int i=0; i<n_records_collect; i++)
        {
                        int collect_result = ADQ_CollectRecord(adq_cu,1, i);
                        unsigned int samples_in_buffer =
MIN(ADQ_GetSamplesPerPage(adq_cu,1), samples_to_collect);

                        if (collect_result)
                        {
                                for (unsigned int j=0; j<samples_in_buffer; j++)
                                {
                                        vector<double>::const_iterator cii;
```

Choose sampling frequency

Immediate restart of timer

Sync mode is sync OFF

Continuous counter

As long as no sync pulse is detected, get timestamp and put result in vector.

If sync pulse active, reset timer immediate restart.

Iterate over result. As ADQ214 gets 2 samples per period, we need to divide time differential by two to get time delta value.
`(*cii+1 - *cii)/2)`

```
                                        vector<double>::const_iterator cii_cal;
                                        for(cii=od_timest.begin(); cii!=od_timest.end(); cii++)
                                        {
                                                for(cii_cal=od_timest_cal.begin(); cii_cal!
=od_timest_cal.end(); cii_cal++)
                                                {
                                                        cout << *cii << endl;
                                                        cout << ((*cii+1 - *cii)/2)<< endl;
                                                        od_timest_cal.push_back((*cii+1 - *cii)/2);
                                                }
                                        }
                                        int dataa = *(data_a_ptr_addr0 + j);
                                        int datab = *(data_b_ptr_addr0 + j);
                                        fprintf(outfileA, "%d\n", dataa);
                                        fprintf(outfileB, "%d\n", datab);
                                }
                                samples_to_collect -= samples_in_buffer;
                        }
                        else
                        {
                                printf("Collect next data page failed!\n");
                                samples_to_collect = 0;
                                i = n_records_collect;
                        }
                }
        }

        // Only disarm trigger after data is collected
        (ADQ_DisarmTrigger(adq_cu,1));

        // Return ADQ to single-trig mode
        (ADQ_MultiRecordClose(adq_cu,1));

}
```