Making Custom Oscilloscope Measurements

Using MATLAB and VB

Updated May 1st, 2017 MH
Signal Path Block Diagram - from Acquisition to Processing

Simplified Oscilloscope Block Diagram
Three Possible Communication Configurations

Configuration 1: Remote Control Scope From Computer

Computational results reside in MATLAB software.
Three Possible Communication Configurations

Configuration 2: MATLAB Software Installed on Scope

Scope Platform

Scope Software

MATLAB

Computational results reside in MATLAB software
Three Possible Communication Configurations

Configuration 3: MATLAB Software Integrated Within Scope Application

MATLAB-assisted dynamic collaboration as the oscilloscope and MATLAB share native capabilities.
Three Possible Communication Configurations

**Configuration 1: Remote Control Scope From Computer**

Scope Platform

Scope App

Computer Platform

MATLAB

GPIB / ENET

Computational results reside in MATLAB software

**Configuration 2: MATLAB Installed on Scope**

Scope Platform

MATLAB

Scope App

Computational results reside in MATLAB software

**Configuration 3: MATLAB Fully Integrated Within Scope Application**

Scope Platform

Scope App

MATLAB

INLINE PROCESSING

MATLAB-assisted DSO shares native capabilities
Workflow Model for Incorporating A New Algorithm

**Traditional Model:**

*Time Scale: 6 - 9 months*

- **End User**
  - Requires a new feature; contacts local representative with request
  - Contacts applications
  - Collects data, logs report, contacts product manager
  - Evaluates request, prioritizes projects
  - Assigns resources, places project in queue
  
- **Sales Engr**
  - Delivered to end user
  - Notifies local rep

- **Appl Engr**
  
- **Prod Mngr**
  - Releases firmware with feature included

- **Engr Mngr**
  
- **Dsn Engr**
  

**Inline Processing Model:**

*Time Scale: Immediate*

- **End User**
  - Incorporates new inline measurement today, and can begin to use it immediately

Proprietary measurements can remain classified and do not need to be shared with scope vendor or the outside world.
How Is A Measurement Selected?

Traditional parameters added to the measurement
How Is A Custom Measurement Selected?

Allows user-defined in-line custom measurements

Custom parameters are added to the measurement list just like traditional parameters.
Fully-Integrated Custom Measurement

Add your own customized measurement or math function

And the result is fully integrated into the scope process as a measurement or trace
Custom MATLAB parameter finds the time elapsed for half-life of the damped sine

The value 3.149 is the number of cycles that have occurred when the signal reaches 50% of its peak amplitude
Real-Time Modification of Custom Measurement

On-scope editor allows loading, saving, and real-time modification of script algorithms.
Arbitrary Waveform Generated on Scope

The damped sinusoid was simulated mathematically
(scope simulates but does not output waveform)

Waveform now exists in scope memory buffer
Peak Decay Ratio

Custom parameter finds the peak decay ratio of the damped sinusoid. The 0.82 reading indicates each peak in the waveform is 0.82 the amplitude of the adjacent previous peak.
Application Example:
MATLAB Digital Filtering
MATLAB Butterworth Filter
MATLAB Butterworth Filter

This MATLAB filter was implemented with only 5 lines of code.

MATLAB Signal Processing Toolbox supports up to 500th order Butterworth filter.
Filter removes slow mean variations by repositioning waveform data
Inverse FFT performed with one line of source code

\[ \text{WformOut} = \text{ifft} (\text{WformIn1} + j \times \text{WformIn2}) ; \]
VB CustomDSO RF Burst Alignment:
Auto-Align Burst Phases and Auto-Scale Burst Amplitudes
Four RF Bursts are acquired. Automatic phase alignment and amplitude matching is needed.
Automatic pulse envelope alignment
Automatic phase alignment
Automatic amplitude scaling adjustment
Automated use of Rescale operator (programmatic)
Phase Alignment Code Sample

```vbc
set XStreamDSO = CreateObject("LeCroy.XStreamDSO")
set app = CreateObject("LeCroy.XStreamDSO")

Set CustomDSO = XStreamDSO.CustomDSO
' CustomDSO ...
CustomDSO.ActionScript1 = "D:\CustomDSO\Starting setup.lss"
CustomDSO.ActionEnable1 = True
CustomDSO.ActionScript2 = "D:\CustomDSO\Set Deskew to Zero.lss"
CustomDSO.ActionEnable2 = True
CustomDSO.ActionScript3 = "D:\CustomDSO\Align envelopes.lss"
CustomDSO.ActionEnable3 = True
CustomDSO.ActionScript4 = "D:\CustomDSO\Align phases.lss"
CustomDSO.ActionEnable4 = True
CustomDSO.ActionScript5 = "D:\CustomDSO\Recalibrate amplitudes.lss"
CustomDSO.ActionEnable5 = True
CustomDSO.ActionScript6 = "D:\CustomDSO\"
CustomDSO.ActionEnable6 = False
CustomDSO.ActionScript7 = "D:\CustomDSO\"
CustomDSO.ActionEnable7 = False
CustomDSO.ActionScript8 = "D:\CustomDSO\"
CustomDSO.ActionEnable8 = False
CustomDSO.Mode = "Basic"
CustomDSO.PresentAtPowerUp = True
CustomDSO.ExecuteAsynchronous = True
CustomDSO.PlugIn1ProgId = "LeCroy.CustomDSODemo.1"
CustomDSO.PlugIn1Installed = False

' Parameter aliasing section
app.measure.P1.alias = "Envel Skew 1-2"
app.measure.P2.alias = "Envel Skew 1-3"
app.measure.P3.alias = "Envel Skew 1-4"
app.measure.P4.alias = "Phase Skew 1-2"
app.measure.P5.alias = "Phase Skew 1-3"
app.measure.P6.alias = "Phase Skew 1-4"
app.measure.P7.alias = "Amplitude 1"
app.measure.P8.alias = "Amplitude 2"
app.measure.P9.alias = "Amplitude 3"
app.measure.P10.alias = "Amplitude 4"


```
VB CustomDSO High Voltage Crossing Detector: Peak Finding Voltage at Iref Crossing
High Voltage Application: Peak Finding Voltages at Iref Crossing
Finds the timestamp of nearest voltage level at first Iref crossing, search all peaks, integer-step through found peaks
High Voltage Application: Peak Finding Voltages at Iref Crossing

Finds the timestamp of nearest voltage level at first Iref crossing, search all peaks, integer-step through found peaks
Threshold and Peak Finding Code Sample

: ******************************************************
': Find the single largest peak current value
: ******************************************************
I0_amps = -9999 ' set to a very large negative number initially
I0_time_array_index = 9999 ' set to a very large number initially
CurrentPeakIndex = TotalNumPeaks ' by default, set the current peak to be
the last peak
LabelText = "Searching for initial peak current.."

app.Acquisition.C1.LabelsText = LabelText
app.Acquisition.C2.LabelsText = LabelText
app.Zoom.Z1.LabelsText = LabelText
app.Zoom.Z2.LabelsText = LabelText

Do While (CurrentPeakIndex > 0) ' loop from right to left, from the rightmost
edge of the waveform toward the cursor position to identify X2max

CurrentPeakIndex = CurrentPeakIndex - 1 ' 

app.Cursors.XPOS1 = CurrentPeakTime

If (app.Cursors.StdCursOfC2.Y1.Result.Value > I0_amps) Then ' mark the
peak
I0_amps = app.Cursors.StdCursOfC2.Y1.Result.Value ' only used to
keep a threshold for the highest value found so far
I0_time_array_index = CurrentPeakIndex
End If

LabelPosition = Cstr(CurrentPeakTime)
app.Acquisition.C1.LabelsPosition = LabelPosition ' set position of labels, based on the array of label
positions defined above
app.Acquisition.C2.LabelsPosition = LabelPosition ' set position of labels, based on the array of label
positions defined above
app.Zoom.Z1.Zoom.HorCenter = LabelPosition
app.Zoom.Z2.Zoom.HorCenter = LabelPosition
app.Cursors.XPOS1 = LabelPosition

LabelText = "Peak found"
app.Acquisition.C1.LabelsText = LabelText
app.Acquisition.C2.LabelsText = LabelText
app.zoom.Z1.LabelsText = LabelText
app.zoom.Z2.LabelsText = LabelText
Dynamic Custom Measurement for Laser Scaling
Dynamic Measurement For Laser Scaling
Dynamic Custom Measurement for Ground Fault Circuit Interruption
Dynamic Measurement Identifies Exact Burst Length and Positions Dynamic Zooms
Application Example:
Linear Regression / Slope Intercept
Oxygen Depletion in a Muscle Fiber
Experiment to measure oxygen depletion in muscle fiber
Nitrogen is injected into container to displace oxygen
Trend line records oxygen level as a function of time.
Long-term trend line can be extracted as a waveform
Long-term trend line of oxygen vs. time
Linear Regression to Determine Slope Intercept of Oxygen Depletion at $t=0$
Application Example:
RF Signal Processing
Hilbert Transform on 10 GHz RF Burst using In-line MATLAB
Hilbert Transform on 10 GHz RF Burst using In-line MATLAB

```matlab
a = WformIn1;
b = hilbert(a);
c = abs(b);
WformOut = c;
```
Application Example:
Decision Process Math Operators
Dynamic Automation Controls

Input Waveform → Static Result

Dynamic Automation Controls

Continuously Sampled Output
Custom scripts can use Automation Controls to act as a feedback mechanism to modify scope settings based on measurement results.

This example uses Automation Controls to automatically modify scope settings and move waveform clear of mask region.

Mask test pass/fail result queried and scope channel vertical offset modified using Automation.

```matlab
h = actxserver('LeCroy.XStreamDSO');
vertoffset = get(h.Acquisition.C2.VerOffset,'Value')
if (h.PassFail.Q1.Out.Result.Value) == 0 %% mask test fails
    vertoffset = get(h.Acquisition.C2.VerOffset,'Value')
    if (vertoffset > 0)  %% trace offset is positive
        set(h.Acquisition.C2.VerOffset,'Value', (vertoffset + 0.005));
    else
        set(h.Acquisition.C2.VerOffset,'Value', (vertoffset - 0.005));
    end
end
```

= automation controls
Measurement on Trace 1 is Gated by Pulses of Trace 2

This example shows how a measurement definition can be dynamically determined by the properties of the waveform data itself. In this example, the portion of data where the measurement takes place on Trace 1 is dependent on the area marked between the two pulses of Trace 2.
Script has not begun computing average because Pass/Fail condition has not yet been True
Script begins averaging F8
Averaging continues on acquisitions only when Pass/Fail condition is True.
Averaging pauses whenever Pass/Fail condition is False
Averaged waveform takes shape of C1 after 287 sweeps. 79 averages have taken place because Pass/Fail condition was True 79 out of 287 sweeps.
Hilbert Transform on Burst using In-line MATLAB

MATLAB Code:

```matlab
1. abs(hilbert(WformIn));
2. output waveform units and scaling with the following
3. Uncomment and calculate as needed
4. 'V', horUnits='S';
5. -0.05; verStep=0.05; horOffset=0; horPerStep=1.0e-10;
```
Getting Started Resources
Sample MATLAB Routines can be downloaded from LeCroy’s Website

- Pre-written math and measurement samples on Teledyne LeCroy’s web site
  www.teledynelecroy.com/MATLAB
Application Note: Implementing MATLAB Filters on Waveform Data

Filter Signals Using MATLAB

Apply MATLAB Based Filters in the DSO’s Processing Path

April 21, 2011

Summary

There is a common need to filter signals prior to analysis. Whether the need is to equalize frequency response or eliminate noise prior to further processing it is very useful to be able to apply a user selected filter to the data.

LeCroy’s XStream™ series oscilloscopes allow users to embed any of MATLAB’s library of filter types right into the oscilloscope’s processing path. In Figure 1, we show an example of a 2-pole, 1 MHz Butterworth low pass filter which has been applied to the acquired waveform using the MATLAB math function.

Figure 1: The response of a MATLAB based 2 pole Butterworth filter (lower trace) to a swept sine input (upper trace).

The MATLAB math function allows the user to call the MATLAB program and execute a MATLAB script file right in the scopes processing path. The output from MATLAB is returned to the next processing stage and operations continue within the scope. Figure 1 shows the basic setup of the MATLAB math function. The function accepts one or two input signals and returns a single output. Selecting the MATLAB tab of the math dialog box allows the user to load an existing .m file or create a new one in the built-in editor, as shown in Figure 2.

Figure 2: A view of the editing window in the WaveMaster MATLAB math function showing part of the MATLAB .m file being viewed.

The .m file used in this example is shown in Figure 3. The filter type used is a Butterworth lowpass filter. MATLAB offers a choice of some 7 filter types. This filter is a relatively slow cutoff second order filter. The command to create the filter coefficients is:

\[(b,a)=butter(2,100/(Fs/2));\]

Where \(b\) represents the numerator coefficients of the digital filter and \(a\) represents the denominator coefficients of the digital filter.

The arguments for the Butterworth filter are order (2 in this case) and the cutoff frequency (this must be normalized to Nyquist which is why we have divided by half of the sampling frequency, \(F_s\)).

The filter is implemented using the filter command:

\[\text{WformOut} = \text{filter}(b,a,\text{WformIn});\]

This applies the filter coefficients to the selected data in this case the input waveform (WformIn).

The following command queries the scope via Microsoft automation to obtain the sampling frequency.

\[\text{Fs} = \text{inSampleRate}();\]

\[\text{in}().\text{Object}[\text{ArQquisition}()].\text{Object}[\text{Item}()];\]

\[
\text{OutResult}: \text{F_s} = 1/\text{Channel}.\text{HorizontalPerStep};
\]
Decoding NRZ Data

Using XDEV Customization to convert to digital waveform

Non-Return to Zero (NRZ) signals use two voltage levels (high and low) to represent logic one and logic zero values. Traditional oscilloscopes can capture and view these waveforms but do not have the capability to decode waveform into digital equivalents. Oscilloscopes with XDEV custom functionality can now decode these waveforms.

In Figure 1, a block of NRZ data is acquired at function F1 and the parameter TIE (Time Interval Error) is applied as parameter B1. The Virtual Clock capability of the TIE reports the underlying binary as 2-48 GHz, which corresponds to the OC-48 datastream. Function F2 allows for a user-defined Matlab script to automatically decode each waveform as shown in Figure 2. The algorithm reads the base frequency from the TIE parameter to locate boundaries of each unit interval. At the algorithm processes each waveform edge, it appends the logic value into a buffer of datastream values. When the loop completes, the decoded waveform and its statistics are displayed as well as recorded to a file. This process will repeat for each acquisition allowing rapid in-line waveform conversion.

Figure 2: The Virtual Clock tab of the TIE-SDK parameter is used to identify the data rate of the pseudo-random bit stream.

Figure 3: Matlab algorithm decodes NRZ data stream.
Using Simulink® Models

Execution of Algorithm Simulation on an X-Stream Scope

Simulink, a software option of MATLAB, is a well known tool to simulate algorithms on a personal computer. In the automotive market, it simulates the vibration of an auto body to optimize its suspension. In communications, it simulates modulation and demodulation of a digital communications system. In the data storage market, it simulates channel elements, such as an equalizer or detector.

Simulation in Simulink usually uses a standard internal signal generator as a signal source, but sometimes circuit simulation with an actual signal is needed. Since LeCroy’s XDEV math option allows X-Stream oscilloscopes to implement MATLAB as an embedded function, Simulink can be implemented as well. One advantage of this method is that simulated results can be analyzed by the advanced analysis functions of the scope, such as jitter and serial data analysis.

Figure 1 shows a simple digital filter model. To implement the Simulink model in the oscilloscope, it must be called from MATLAB where it is stored as “Model1”. Figure 2 shows the setup for mathematic F1, using MATLAB as an embedded function. To call the Simulink model from MATLAB, the “tm” command is used.

command is used.

The syntax of the command is as follows:

```matlab
[t, y, Model1] = sim('ModelName', SimulinkTime, Options, DataArray)
```

“DataArray”, the input data for Simulink Model, must be a two-dimensional array consisting of time and voltage values. Since the waveform data, Wform1n, transferred from the oscilloscope to MATLAB is a one-dimensional array of voltage values, it must be converted to a two-dimensional array in MATLAB with the addition of the time data array. To make time data, MATLAB needs the following information: time resolution and number of points of Wform1n. The time resolution can be read as “TmainPerPoint” from the oscilloscope. Since the X-Stream oscilloscope uses Windows-based software, it is COM (Command Object Model) based, and the needed parameters can be read through ActiveX. The MATLAB command for this operation is shown below:

```matlab
SimulinkTime = t
Options = SimulationTarget
DataArray = Wform1n
[t, y, Model1] = sim('ModelName', SimulinkTime, Options, DataArray)
```

And the number of points of the sample length of Wform1n is obtained by using the “size” command. So now the duration of Wform1n can be calculated as the product of time resolution and number of points.

Then time array “t” is generated using these parameters. The data array “y” is made from the calculated time array and the Wform1n amplitude by concatenation. Then “y” is sent to the Simulink model by the “sim” command. In this case, the Simulink model name is “Model1”, under which the Simulink model was stored previously. Since simulation time must match the duration of Wform1n, the variable “Duration” is used in the command.
Recommended Coding Methodology

Given the complexity of MATLAB’s syntax for handling multi-tiered automation objects, it is highly recommended to create object variables at each level of the hierarchy. In MATLAB, this kind of variable is referred to as a handle. This simplifies the code, makes it more readable and easier to debug. Here are some examples of object variables that can be directly used in your script for referencing App.Acquisition.C1.Out.Result, App.Measure.P1.Out.Result and App.Math.F1.Out.Result.

```matlab
% Instantiate of Scope Application object at the ID adress referenced.
% (Use 127.0.0.1 when running MATLAB on the scope)
app = actxserver(‘LeCroy.XStreamUSB’); % ’172.16.0.1.55’

% creation of object variables 1 level down from top-level
acq = app.Object.Item(‘Acquisition’);
math = app.Object.Item(‘Math’);
meas = app.Object.Item(‘Measure’);
FF = app.Object.Item(‘FastFill’);

% creation of object variables 1 level further
cl = acq.Object.Item(‘C1’);
fl = math.Object.Item(‘F1’);
p1 = meas.Object.Item(‘P1’);
p1Operator = p1.Object.Item(‘Operator’);

% creation of object variables to results
cl_results = cl.Out.Result;
fl_results = fl.Out.Result;
p1_results = p1.Out.Result;
```

Frequently Used Code Blocks

This section includes blocks of code that are frequently used. 

A) Accessing channel results

```matlab
% drill into hierarchy to C1 results
app = actxserver(‘LeCroy.XStreamUSB’); %’172.16.0.1.55’
acq = app.Object.Item(‘Acquisition’);
cl = acq.Object.Item(‘C1’);
cl_results = cl.Out.Result;

% Get entire data array
cl_data = get(cl_results, ’DataArray’);

% Get C1 properties
cl_ClientTemp = cl_results.ClientTemp;
cl_status = cl_results.Status;
cl_NumSweeps = cl_results.NumSweeps;
cl_VpsdStep = cl_results.VpsdStep;
```

B) Accessing Parameter results

```matlab
% Drill into hierarchy to P1 results
app = actxserver(‘LeCroy.XStreamUSB’);
meas = app.Object.Item(‘Measure’);
p1 = meas.Object.Item(‘P1’);
p1_results = p1.Out.Result;
p1_meanresults = p1.Object.Item(′mean′);
p1_numresults = p1.Object.Item(′num′);
```

Here are examples of getting and setting properties of the VerScale CVAR for channel 1:

```matlab
% read the value of the VerScale CVAR.
CLVDiv = get(cl.Item(′VerScale′), ′Value′)

% read the Range and Type properties of the VerScale CVAR
range = get(cl.Item(′VerScale′), ′Range′)
type = get(cl.Item(′VerScale′), ′Type′)

% set VerScale to 0.789 V/div
set(cl.Item(′VerScale′), ′Value′, 0.789)
```
Making Custom Oscilloscope Measurements

Questions?