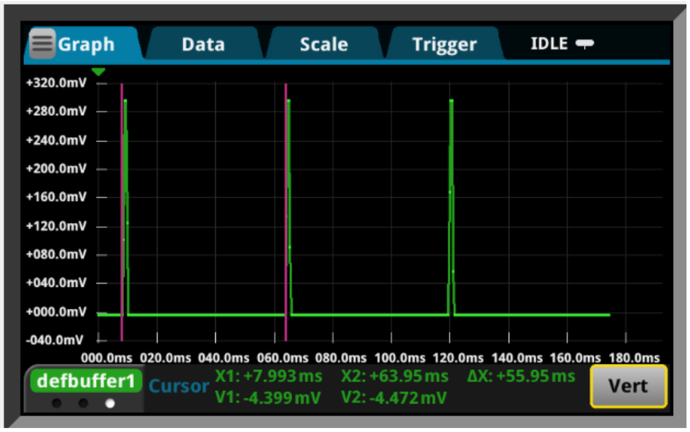
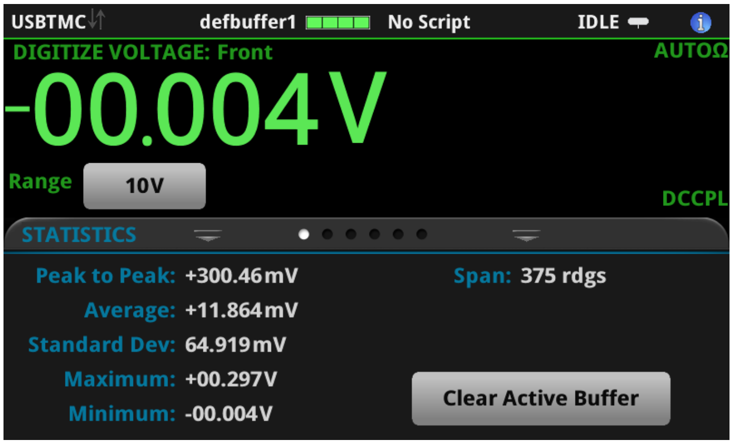
Use Digitize Voltage Feature to Obtain Frequency of a Repetitive Signal

If you have some guess about the expected Frequency, you can use digitizer to acquire N samples so that you know you have at least two peaks captured.



Cursor X1: +7.993 ms X2: +63.95 ms ΔX: +55.95 ms
V1: -4.399 mV V2: -4.472 mV

From the buffer statistics, we know the maximum voltage in the waveform.



DIGITIZE VOLTAGE: Front

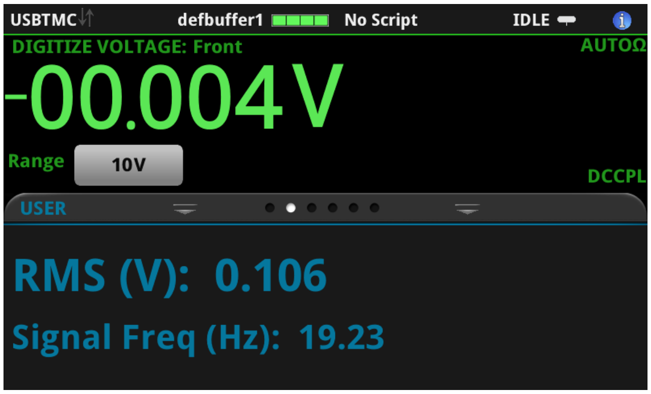
-00.004 V

Range 10V

Peak to Peak: +300.46 mV
Average: +11.864 mV
Standard Dev: 64.919 mV
Maximum: +00.297 V
Minimum: -00.004 V
Span: 375 rdgs

Use that maximum voltage as a hint to a peak detector algorithm written in the Lua based TSP scripting.

In this case, we are also displaying RMS from the peak-to-peak voltage info.



DIGITIZE VOLTAGE: Front

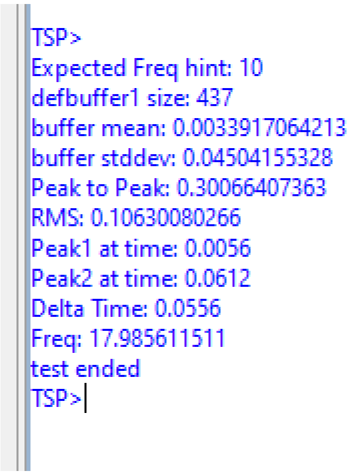
-00.004 V

Range 10V

RMS (V): 0.106

Signal Freq (Hz): 19.23

If I run the code from Test Script Builder with debug_flag set to true, we see:



```
TSP>
Expected Freq hint: 10
defbuffer1 size: 437
buffer mean: 0.0033917064213
buffer stddev: 0.04504155328
Peak to Peak: 0.30066407363
RMS: 0.10630080266
Peak1 at time: 0.0056
Peak2 at time: 0.0612
Delta Time: 0.0556
Freq: 17.985611511
test ended
TSP>
```

The code:

```
--[[

  use digitize V on DAQ6510, DMM6500 or DMM7510

  Goal:  be able to report the AC waveform freq

  Method:
     Acquire a little more than one period worth of data.
     Find timestamps corresponding to the two peaks/max values.
     Freq = 1/(t2-t1)


  Tested with KEITHLEY INSTRUMENTS,MODEL DAQ6510,04448301,1.7.12b
  and AFG31102 for simulating test signals

]]--


function indexOf(meas_values, target)

    for i, v in ipairs(meas_values) do
      if v >= target then
        return i
      end  -- if
    end  -- for loop
    return nil  -- if target not found

end  -- function


function findPeaks(maxReading)

          percent_of_max = 0.9
          -- after first peak found, skip forward into buffer for second search start point
          -- case these are squarewave, do not want to find secong peak too soon!
          sample_offset = 0.25 * defbuffer1.n

          -- copy defbuffer1 to a table so we can pass it to function
          voltages = {}
          for i = 1, defbuffer1.n do
            voltages[i] = defbuffer1.readings[i]
          end  -- for loop

          peak_idx1 = indexOf(voltages, percent_of_max * maxReading)
          --print("First Peak Index : "..peak_idx1)

          -- copy a subset of defbuffer1 to search for second peak
          voltages2 = {}
          for i = 1, (defbuffer1.n - peak_idx1-sample_offset) do
            voltages2[i] = defbuffer1.readings[i+peak_idx1+sample_offset]
          end  -- for loop

          peak_idx2 = indexOf(voltages2, percent_of_max * maxReading)
          -- index of voltage2 table not same as index of defbuffer1
          -- correct our index to be that of defbuffer1 so we can get timestamp at peak2 buffer index
          peak_idx2 = peak_idx2 + peak_idx1 + sample_offset
          --print("Second Peak Index : "..peak_idx2)

          peak1 = defbuffer1.relativetimestamps[peak_idx1]
          peak2 = defbuffer1.relativetimestamps[peak_idx2]
          time_between_peaks = defbuffer1.relativetimestamps[peak_idx2] - defbuffer1.relativetimestamps[peak_idx1]

          return peak1, peak2, time_between_peaks

end  -- function


function config_digitizer(sample_rate, buffer_size, meas_range)
          --Set the measurement function to Digitize Voltage to capture the power-up behavior
          dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
          --Voltage range must be fixed when using Digitizing Voltage
          dmm.digitize.range= meas_range
          dmm.digitize.samplerate = sample_rate
          dmm.digitize.aperture = dmm.APERTURE_AUTO
          --Changing count is optional.  The reading buffer capacity is the determining factor
          --dmm.digitize.count = 1
          --Set the input impedance to auto so it select 10G for the 10V range
          dmm.digitize.inputimpedance = dmm.IMPEDANCE_AUTO

          --Set the buffer size to number of samples to capture
          -- each trigger model acquisition, will acqure this many samples and then stop
          defbuffer1.clear()
          defbuffer1.capacity = buffer_size

          -- create a very simple trigger model
          blockNumber = 1
          trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR)

          blockNumber = blockNumber + 1
          trigger.model.setblock(blockNumber,trigger.BLOCK_DIGITIZE, defbuffer1, buffer_size)

end  -- function

function meas_freq(debug_flag)
          -- if configured, just call this to run it
          trigger.model.initiate()
          --Waits for the trigger model to finish collecting data before proceeding
          waitcomplete()

          -- refresh our stats
          stats = buffer.getstats(defbuffer1)
          t1, t2, deltaT = findPeaks(stats.max.reading)  -- use max value as hint
          pk_pk=stats.max.reading-stats.min.reading      -- compute peak to peak
          rms=pk_pk/(2*math.sqrt(2))                   -- compute RMS

          if debug_flag == true then
            print()
            print("Expected Freq hint: "..Expected_Freq)
                  print("defbuffer1 size: "..stats.n)
                  print("buffer mean: "..stats.mean)
                  print("buffer stddev: "..stats.stddev)

                  print("Peak to Peak: "..pk_pk)
                  print("RMS: "..rms)

                  print("Peak1 at time: "..t1)
                  print("Peak2 at time: "..t2)
                  print("Delta Time: "..deltaT)
                  print("Freq: "..1/deltaT)
          end
```

```
end -- function

-- *******************************************************
--
--   Main Code here
--
--
-- *******************************************************

Expected_Freq = 10
SampleDuration = 1.75 * 1/Expected_Freq  -- sample longer than one period of the waveform
SampleRate = 2.5e3  -- can be 1KHz to 1MHz
numofsamples = SampleDuration * SampleRate  -- stay within max size for defbuffer1
dcv_meas_range = 10



reset()
eventlog.clear()


config_digitizer(SampleRate, numofsamples, dcv_meas_range)


display.changescreen(display.SCREEN_GRAPH_SWIPE)
REPEAT = 1
for j = 1, REPEAT do  -- in case you want it to run more than once

        --meas_freq(debug_flag)  -- debug_flag will print some info back to instrument console
        meas_freq(false)   -- true or false

        display.clear()

        display.settext(display.TEXT1, string.format("RMS (V):  %.3f",rms))
        display.settext(display.TEXT2, string.format("Signal Freq (Hz):  %.2f",1/deltaT))
        if j == 1 then display.changescreen(display.SCREEN_USER_SWIPE) end
        --display.changescreen(display.SCREEN_GRAPH_SWIPE)

        --delay(1)  -- loop delay if you want

end  -- loop on j to run this N times

print("test ended")
```