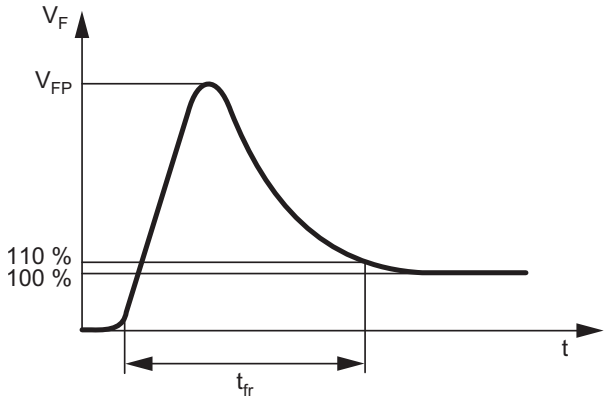
# Turn-On Time of TVS Diodes


http://www.hppi.de/

March 10, 2021

# Forward Turn-On Time

Forward recovery time $t_{fr}$ [1]: The time required for the voltage to reach a specified value (normally 110 % of the steady state forward voltage drop), after instantaneous switching from zero or a specified reverse voltage to a specified forward biased condition (forward current)




Ref: [1]

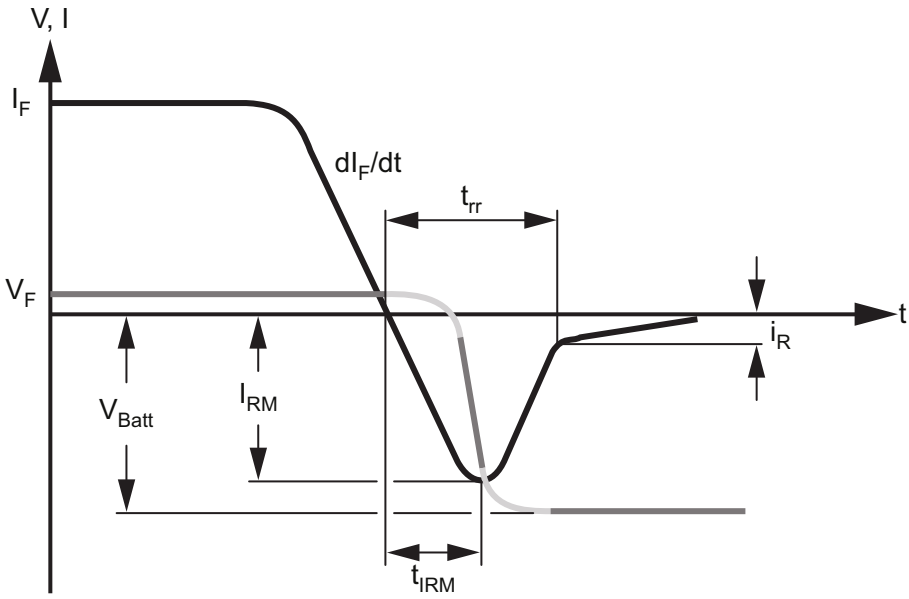
# Reverse Turn-On Time

Reverse recovery time $t_{rr}$ [1], [2]: The time required for the current to reach a specified reverse current, $i_R$ (normally 25 % of $I_{RM}$), after switching from a specified forward current $I_F$ to a specified reverse biased condition (reverse voltage $V_{Batt}$) with a specified slope $dI_F/dt$

Reverse turn-on (overshoot recovery) time $t_{on}$: The time required for the reverse voltage to reach a specified value after overshoot (normally 110 % of the steady state reverse voltage drop), after instantaneous switching from zero forward current to a specified reverse current biased condition (reverse current $I_R$)

- The reverse turn-on (overshoot recovery) time $t_{on}$ is evaluated in this report

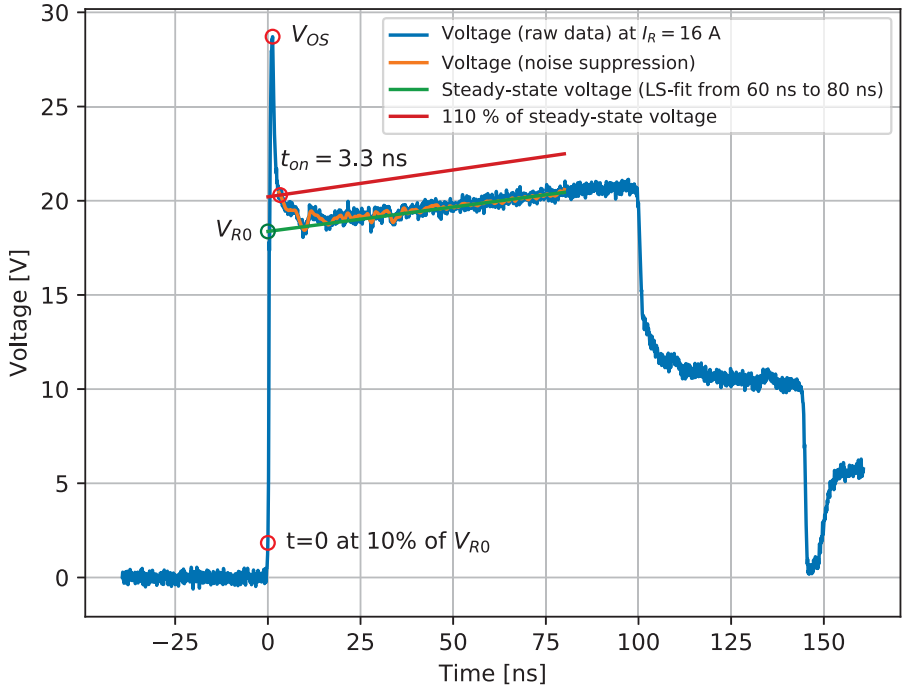
# Reverse Recovery Time $t_{rr}$




Ref: [1]

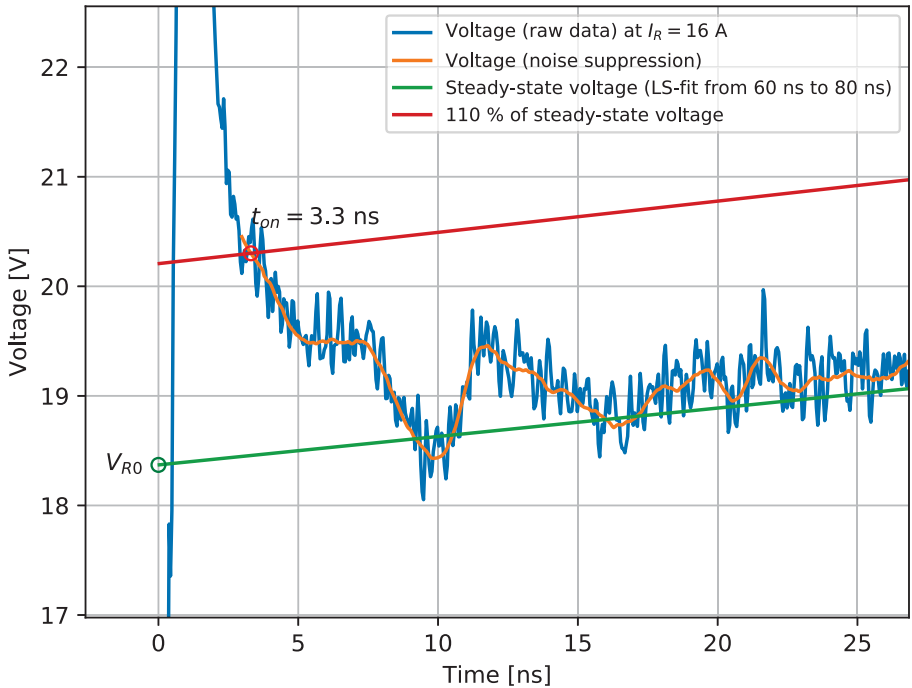
# Reverse Turn-On (Overshoot Recovery) Time $t_{on}$

## Example: Reverse Overshoot Voltage $V_{os}$



# Reverse Turn-On (Overshoot Recovery) Time $t_{on}$

## Example: Zoom View

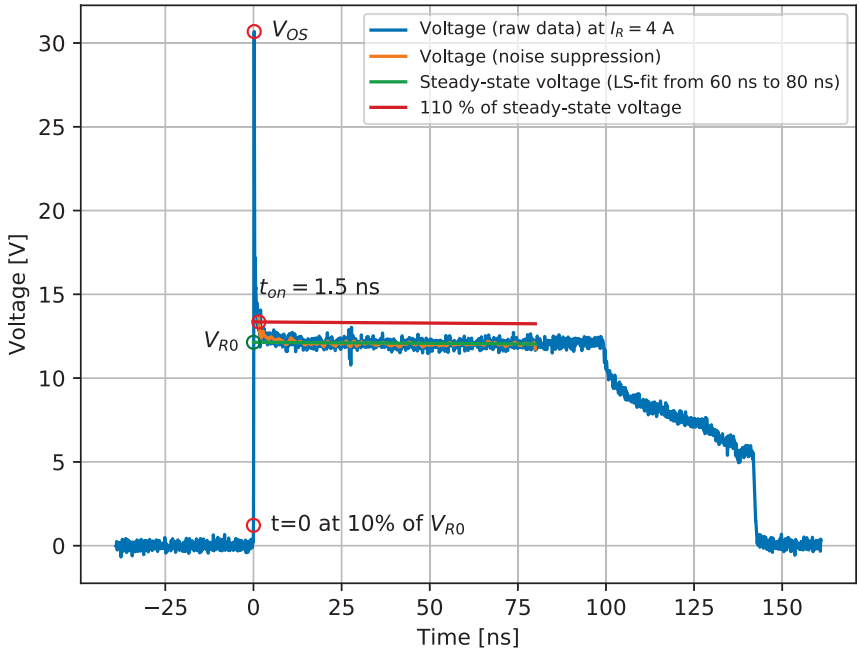


# Reverse Turn-On (Overshoot Recovery) Time $t_{on}$

## Procedure: How To Extract $t_{on}$

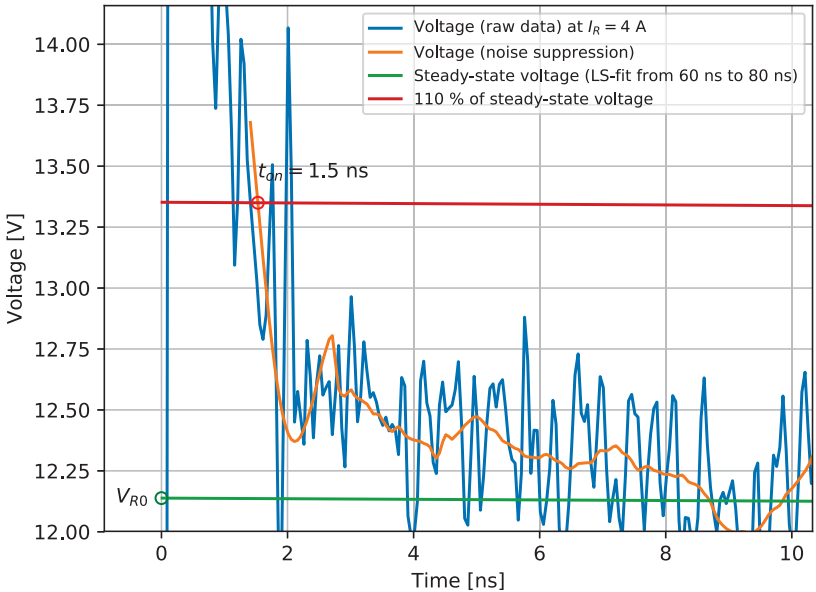
1. Evaluate steady-state voltage by LS-fit from t1=60 ns to t2=80 ns
   - t1 and t2 should be defined for proper steady-state
2. Calculate steady-state voltage $V_{R0}$ at $t = 0$
3. Deskew time axis at $t = 0$ at 10 % of $V_{R0}$
4. Calculate 110 % of the steady-state voltage
   (LS-fit from 60 ns to 80 ns)
5. Apply appropriate noise suppression of the voltage raw data
6. Calculate intersect point $t_{on}$ of filtered voltage raw data and 110 % of the steady-state voltage after overshoot

# D5V0F1U2LP3 [3], $I_R = 4$ A at 100 ps Rise Time

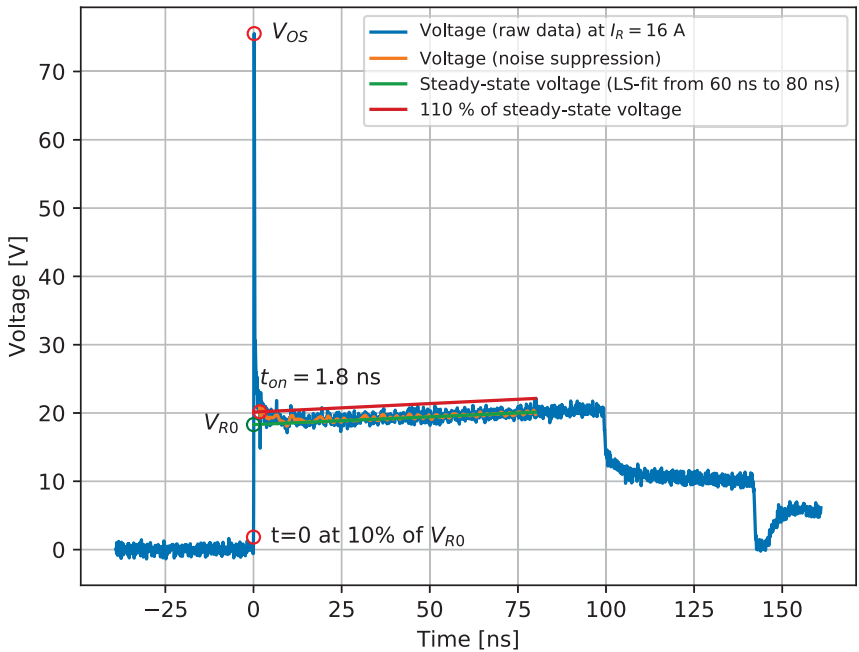


# D5V0F1U2LP3 [3], $I_R = 4$ A at 100 ps Rise Time

## Detail View

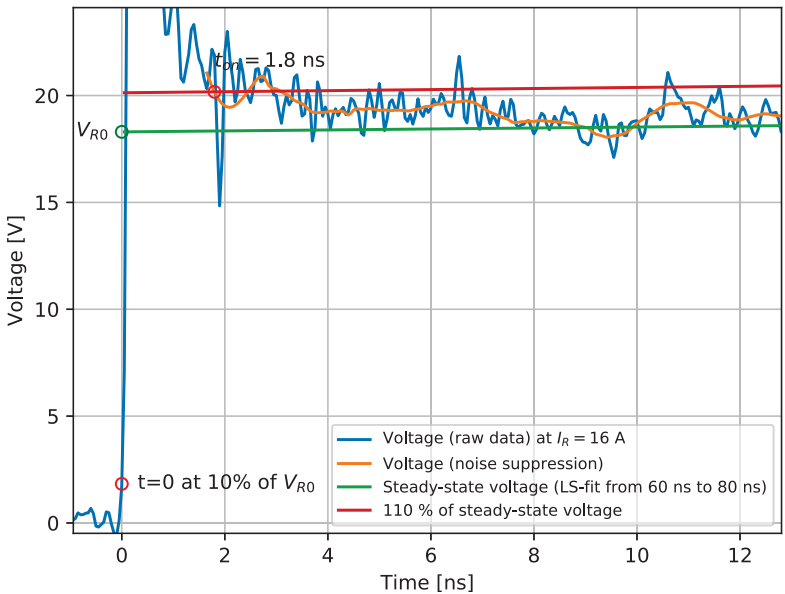


# D5V0F1U2LP3 [3], $I_R = 16$ A at 100 ps Rise Time

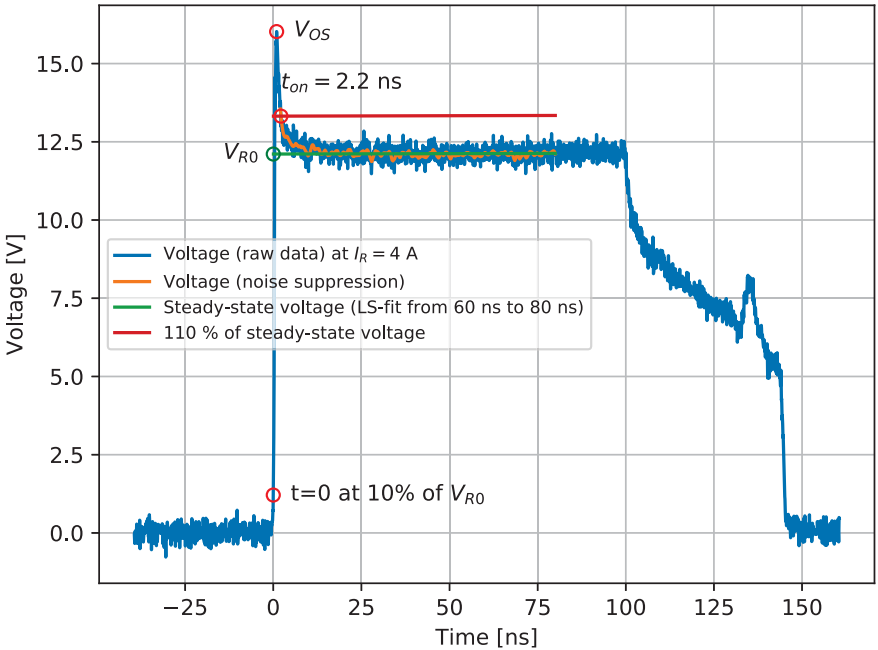


# D5V0F1U2LP3, $I_R = 16$ A at 100 ps Rise Time

## Detail View

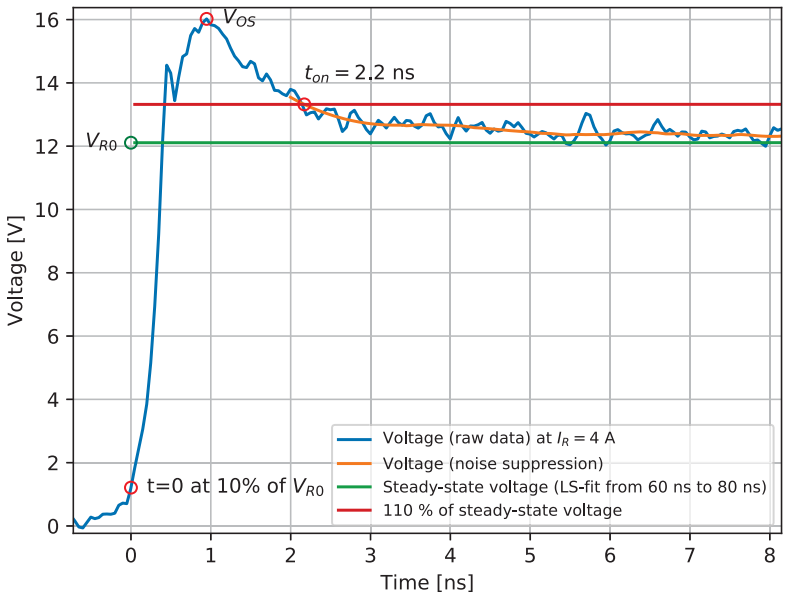


# D5V0F1U2LP3 [3], $I_R = 4$ A at 1 ns Rise Time

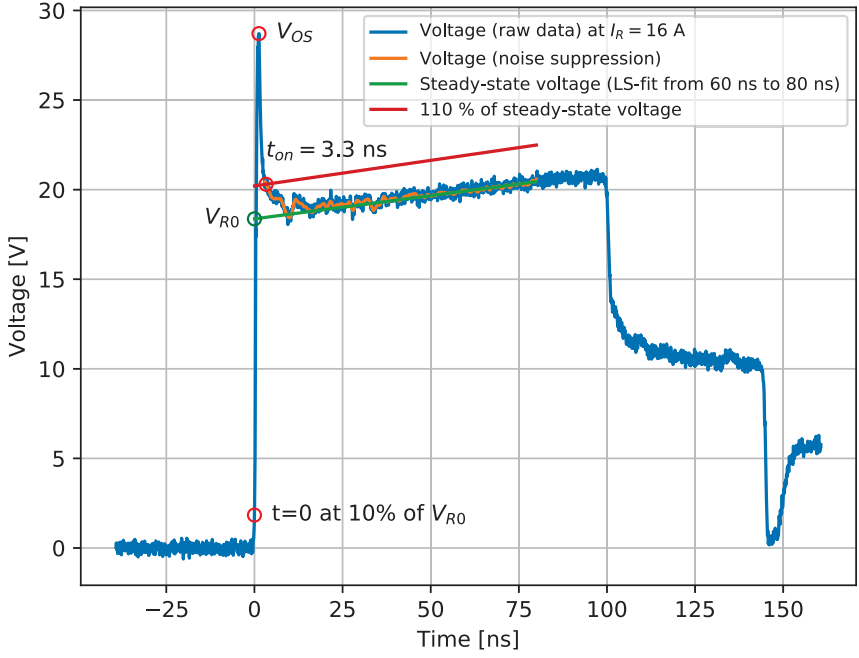


# D5V0F1U2LP3, $I_R = 4$ A at 1 ns Rise Time

## Detail View

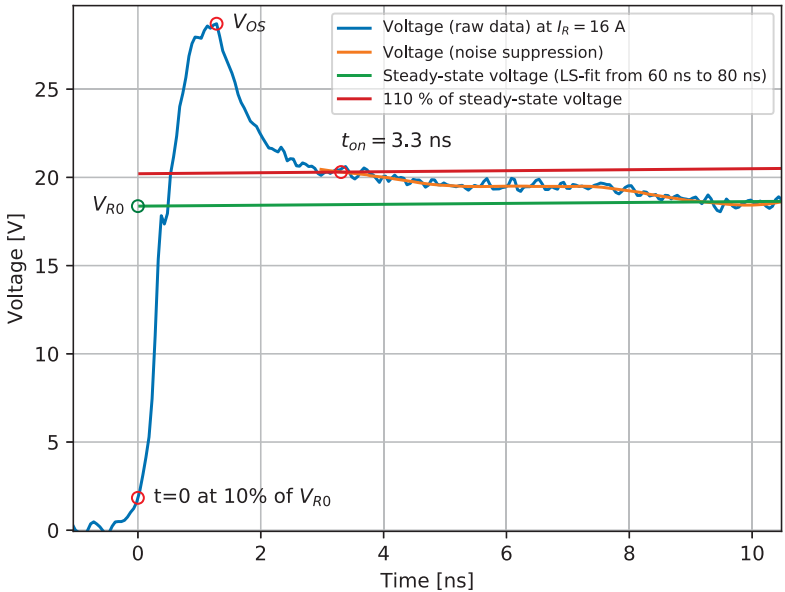


# D5V0F1U2LP3 [3], $I_R = 16$ A at 1 ns Rise Time

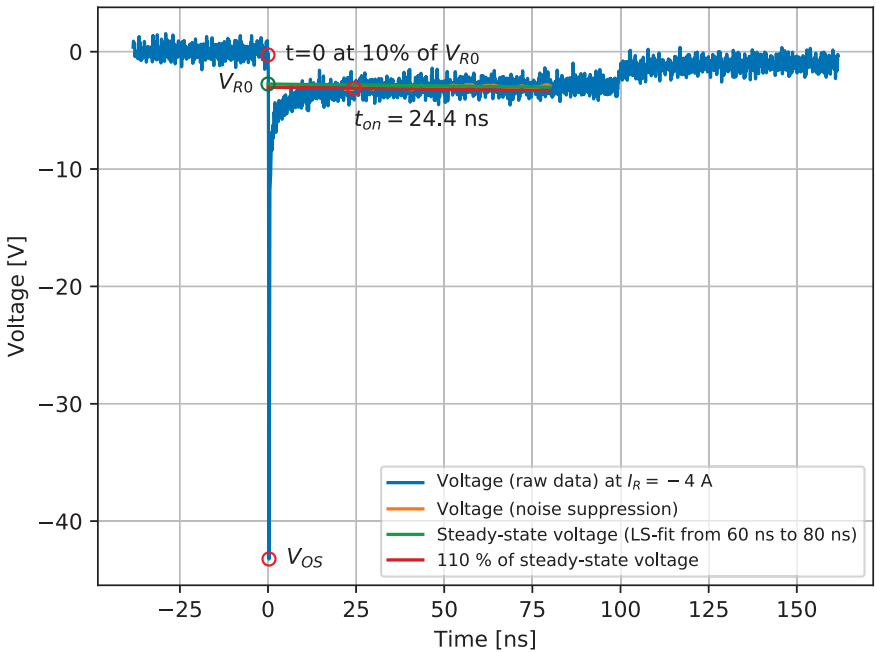


# D5V0F1U2LP3, $I_R = 16$ A at 1 ns Rise Time

## Detail View

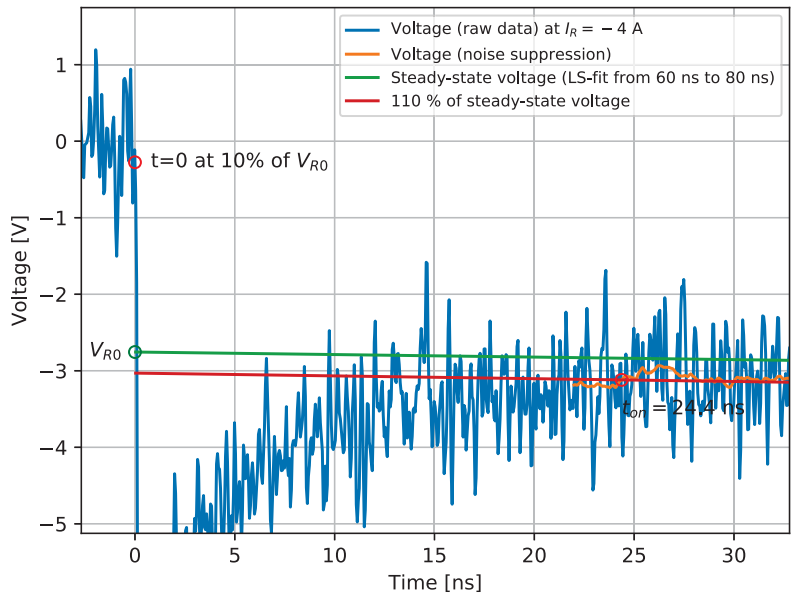


# D5V0F1U2LP3 [3], $I_R = -4$ A at 100 ps Rise Time

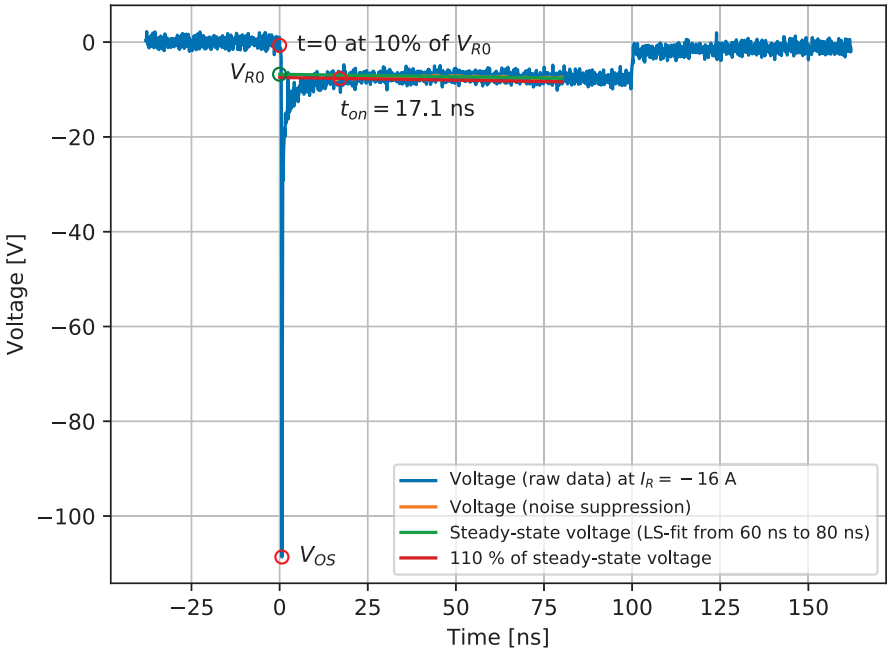


# D5V0F1U2LP3, $I_R = -4$ A at 100 ps Rise Time

## Detail View

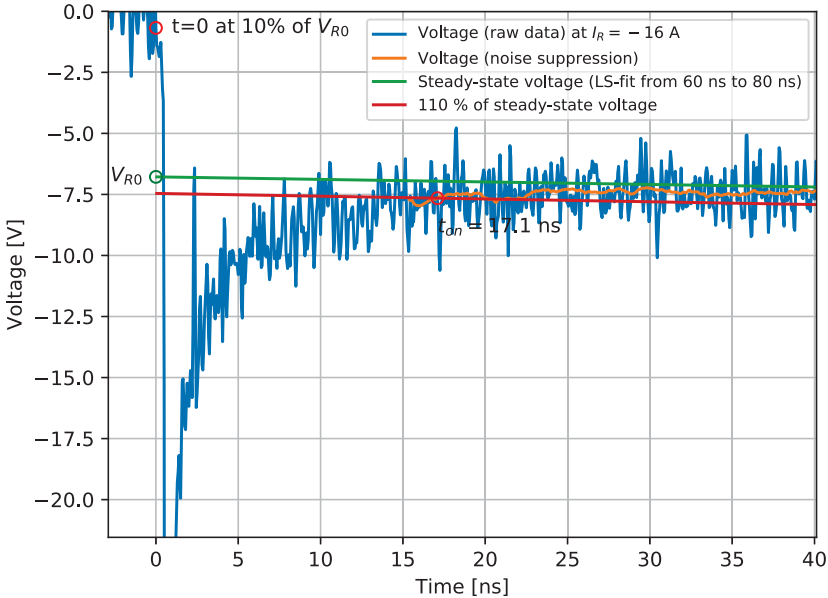


# D5V0F1U2LP3 [3], $I_R = -16$ A at 100 ps Rise Time

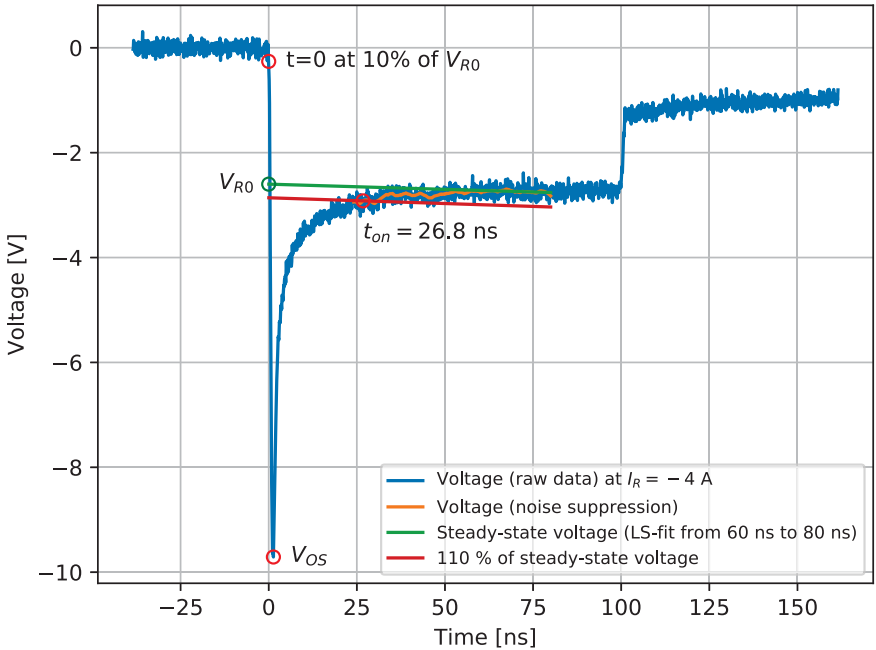


# D5V0F1U2LP3 [3], $I_R = -16$ A at 100 ps Rise Time

## Detail View

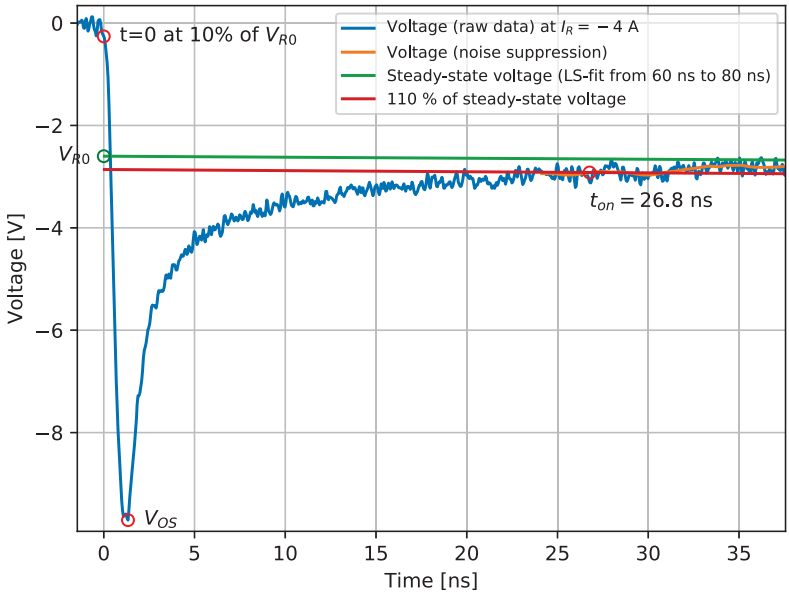


# D5V0F1U2LP3 [3], $I_R = -4$ A at 1 ns Rise Time

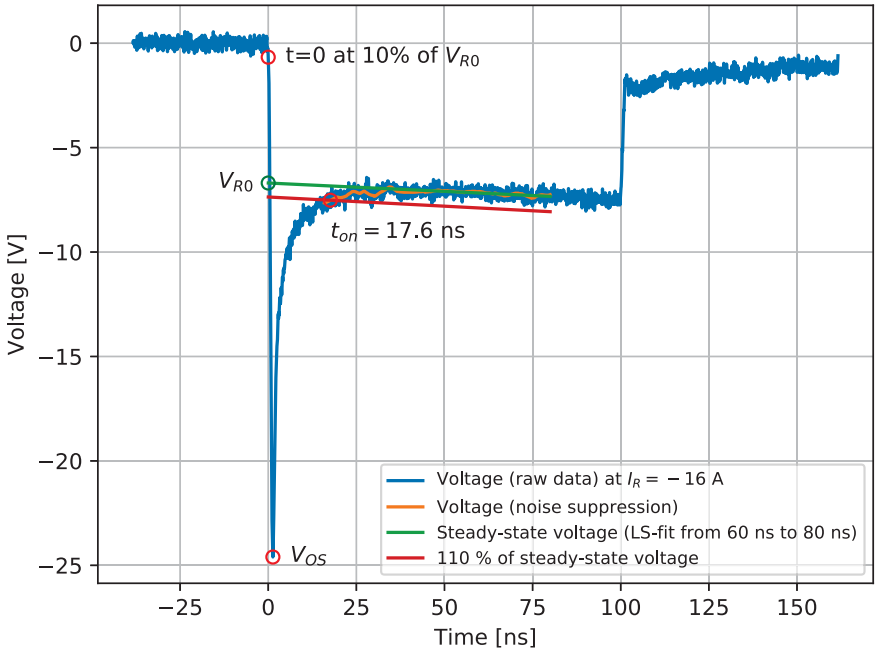


# D5V0F1U2LP3 [3], $I_R = -4$ A at 1 ns Rise Time

## Detail View

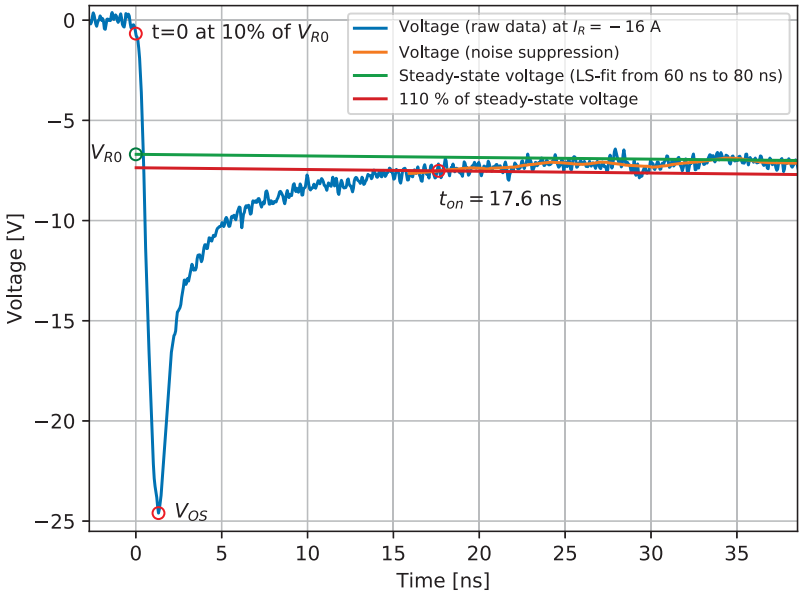


# D5V0F1U2LP3 [3], $I_R = -16$ A at 1 ns Rise Time

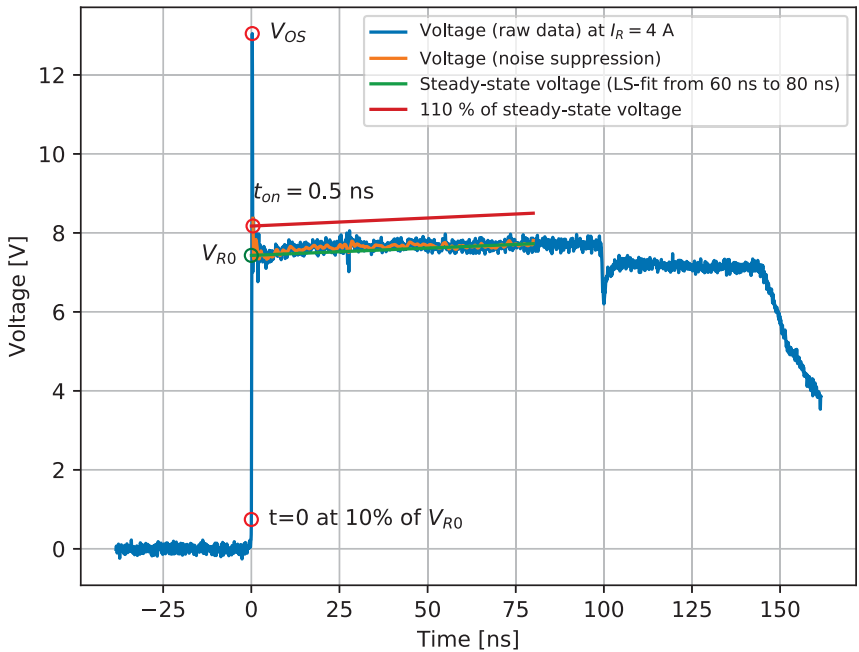


# D5V0F1U2LP3 [3], $I_R = -16$ A at 1 ns Rise Time

## Detail View

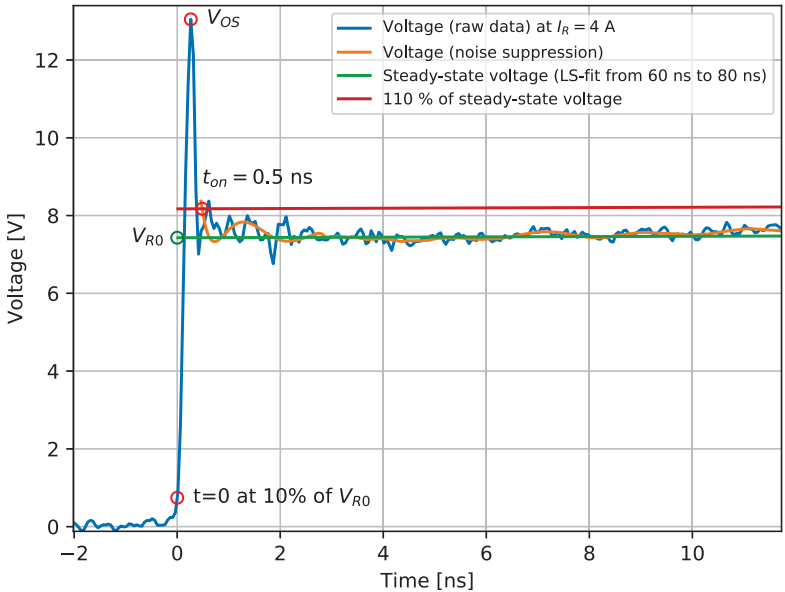


# D5V0M1U2LP3 [4], $I_R = 4$ A at 100 ps Rise Time

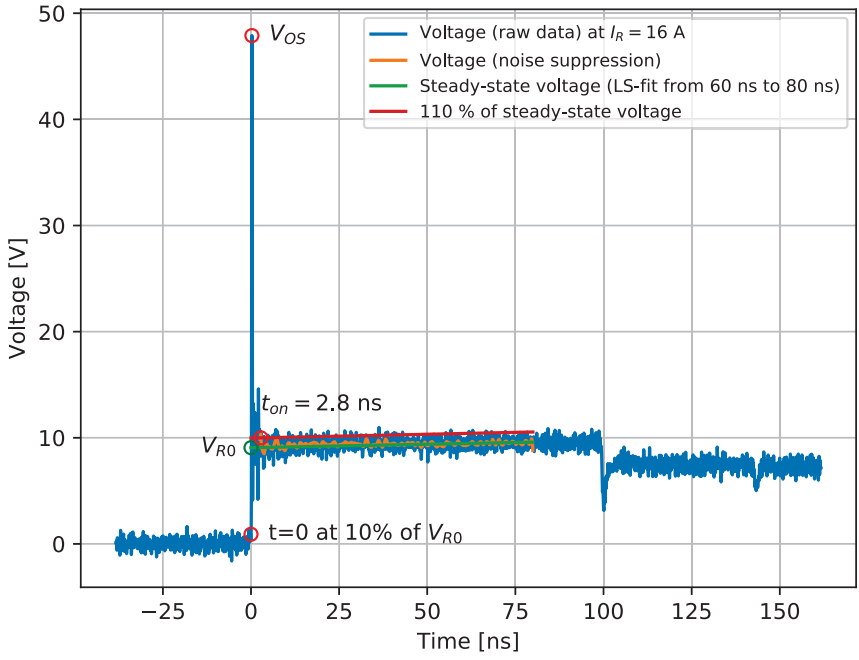


# D5V0M1U2LP3 [4], $I_R = 4$ A at 100 ps Rise Time

## Detail View

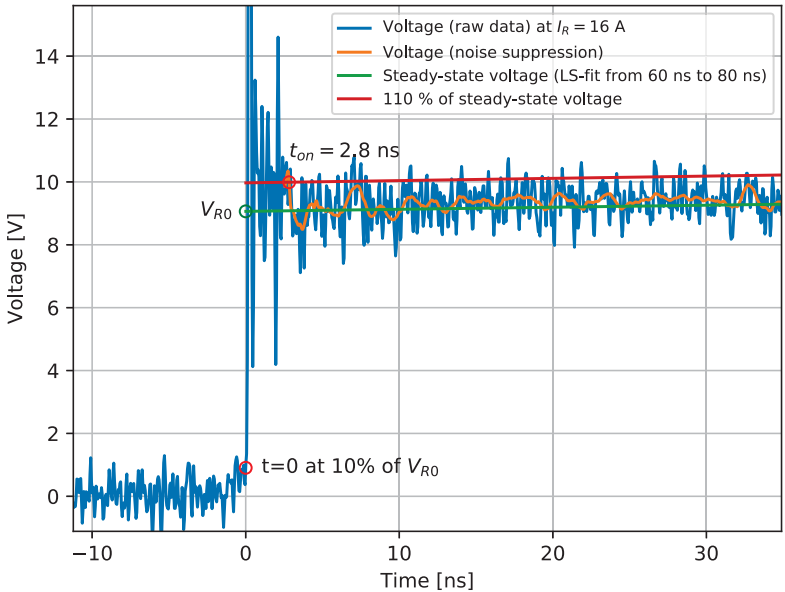


# D5V0M1U2LP3 [4], $I_R = 16$ A at 100 ps Rise Time

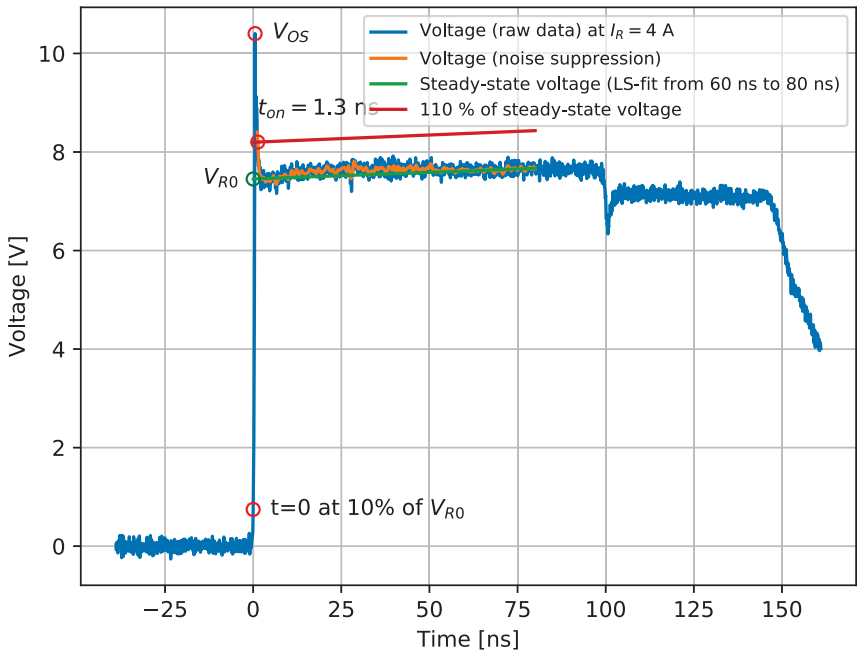


# D5V0M1U2LP3 [4], $I_R = 16$ A at 100 ps Rise Time

## Detail View: Strong Resonant Ringing

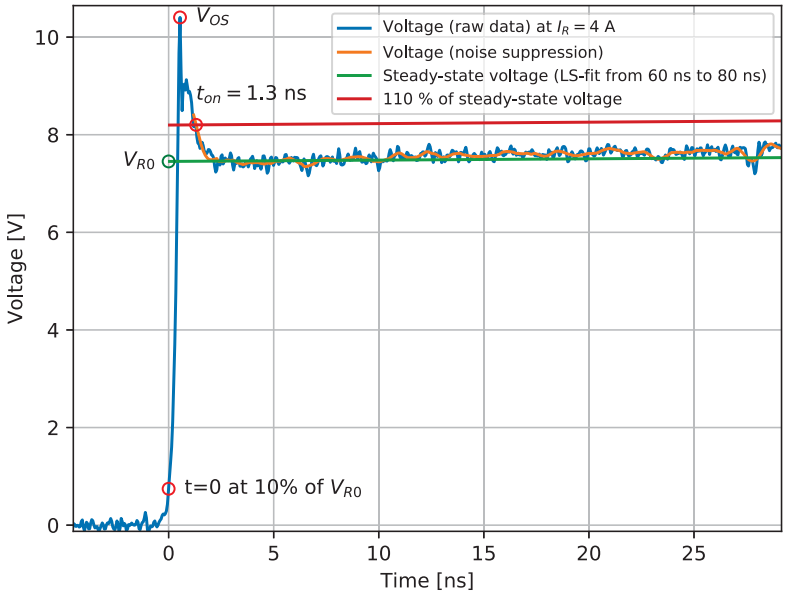


# D5V0M1U2LP3 [4], $I_R = 4$ A at 1 ns Rise Time

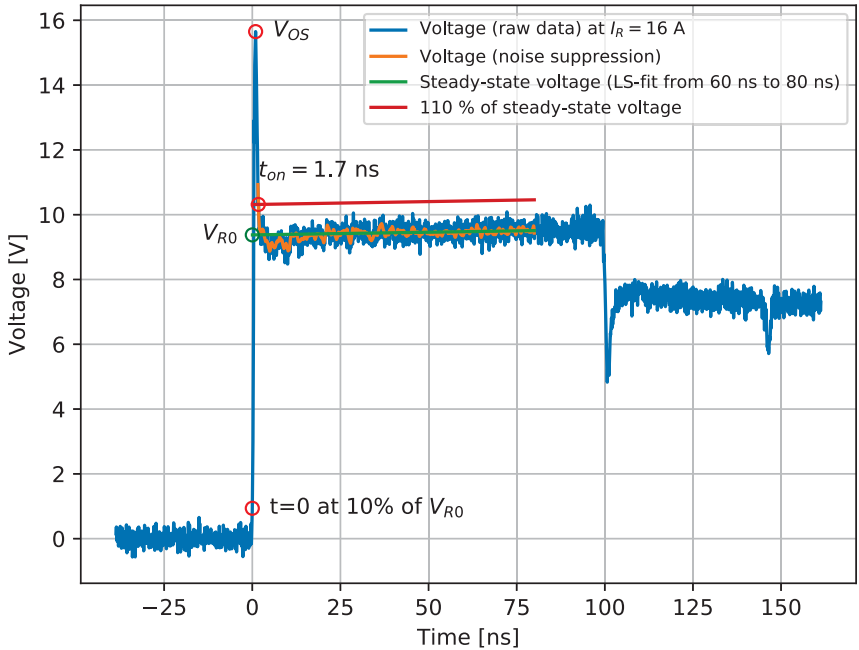


# D5V0M1U2LP3 [4], $I_R = 4$ A at 1 ns Rise Time

## Detail View

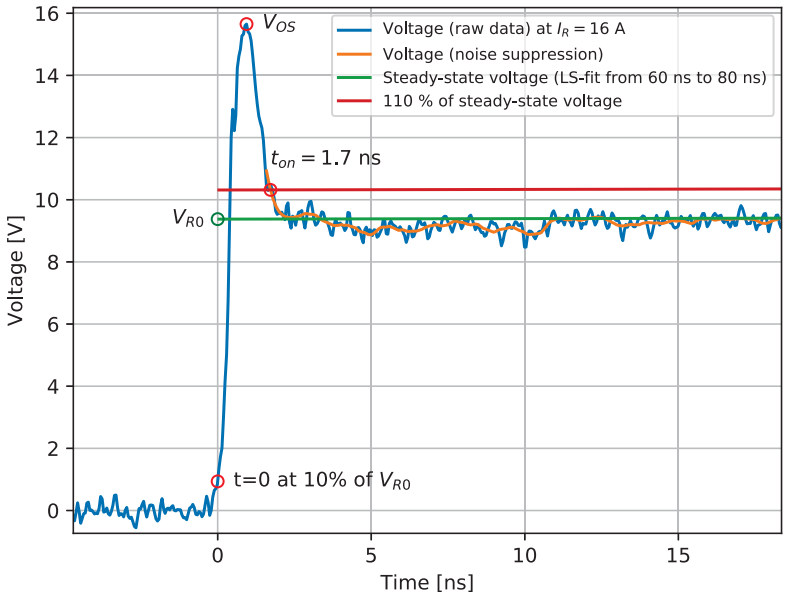


# D5V0M1U2LP3 [4], $I_R = 16$ A at 1 ns Rise Time

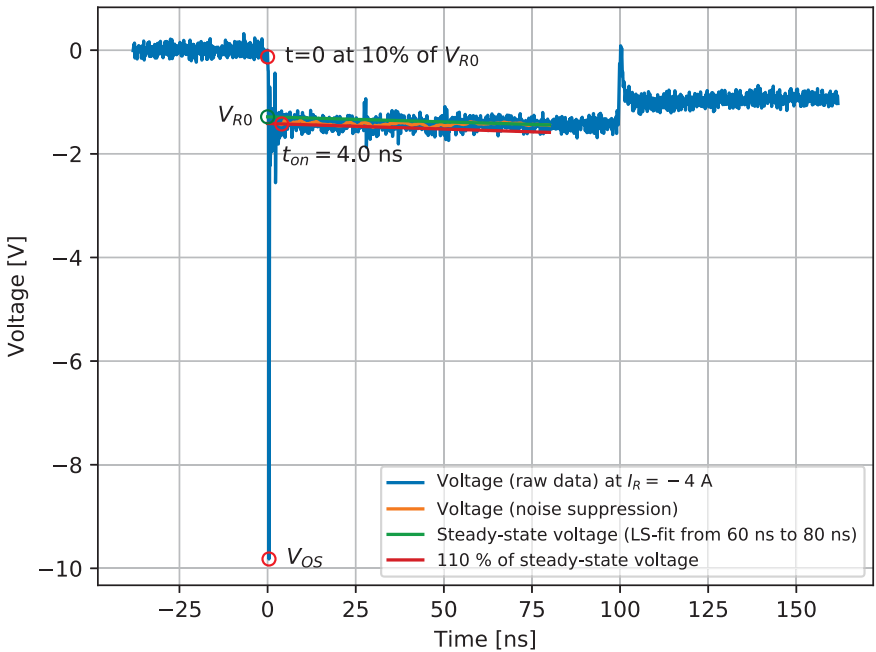


# D5V0M1U2LP3 [4], $I_R = 16$ A at 1 ns Rise Time

## Detail View

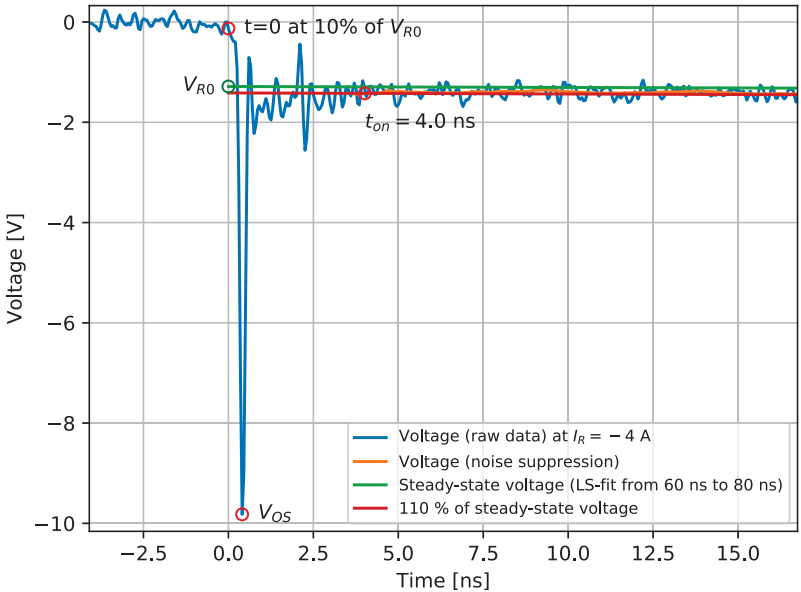


# D5V0M1U2LP3 [4], $I_R = -4$ A at 100 ps Rise Time

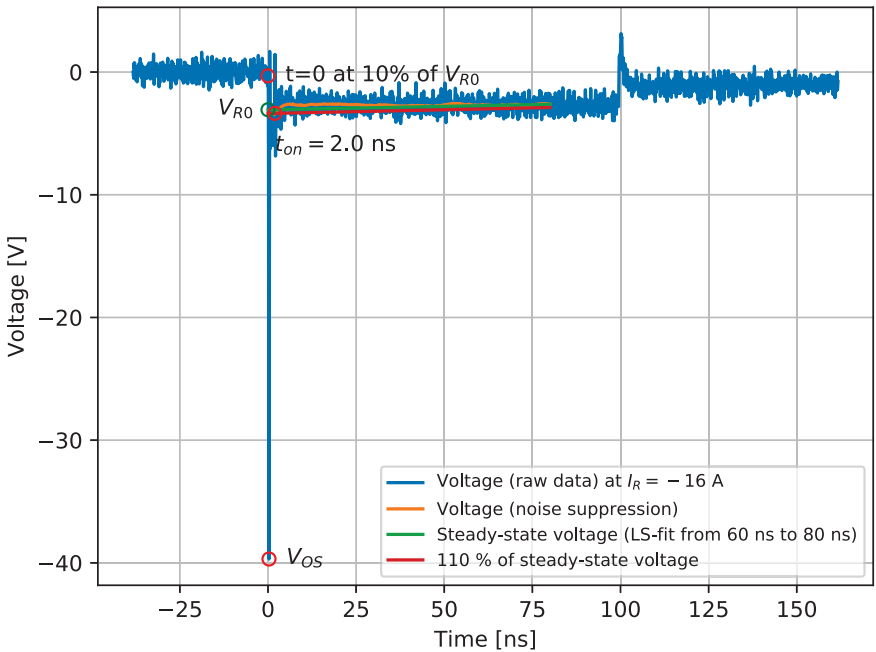


# D5V0M1U2LP3 [4], $I_R = -4$ A at 100 ps Rise Time

## Detail View

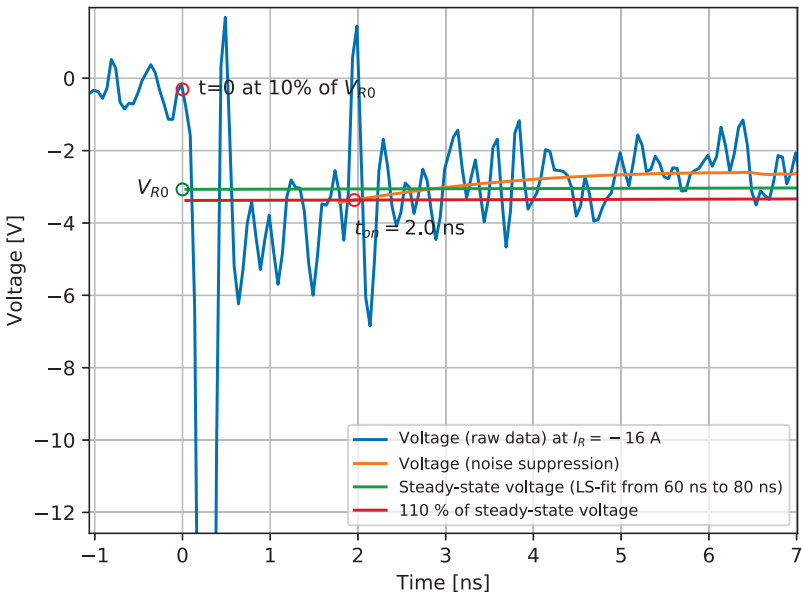


# D5V0M1U2LP3 [4], $I_R = -16$ A at 100 ps Rise Time

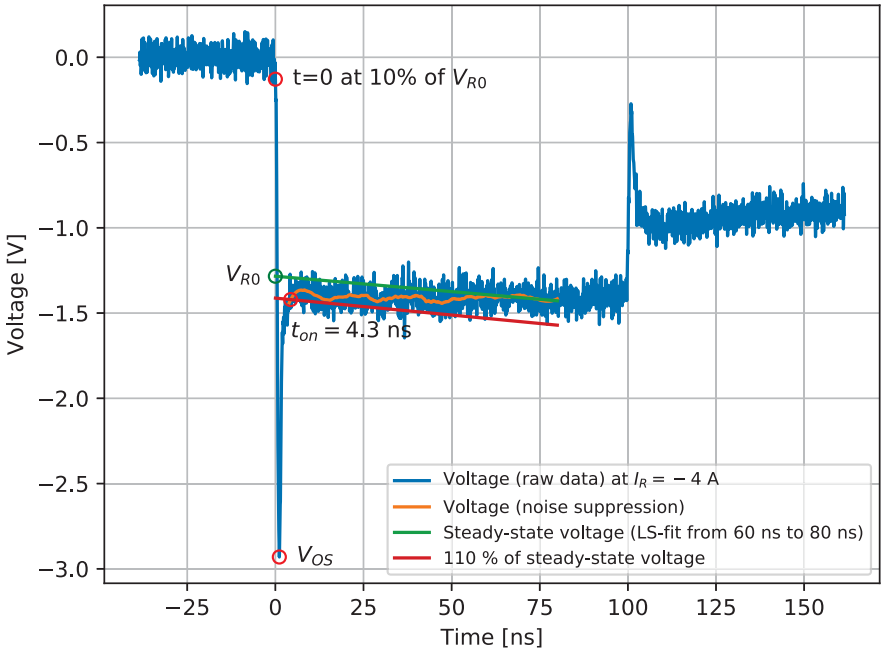


# D5V0M1U2LP3 [4], $I_R = -16$ A at 100 ps Rise Time

## Detail View: Strong Resonant Ringing

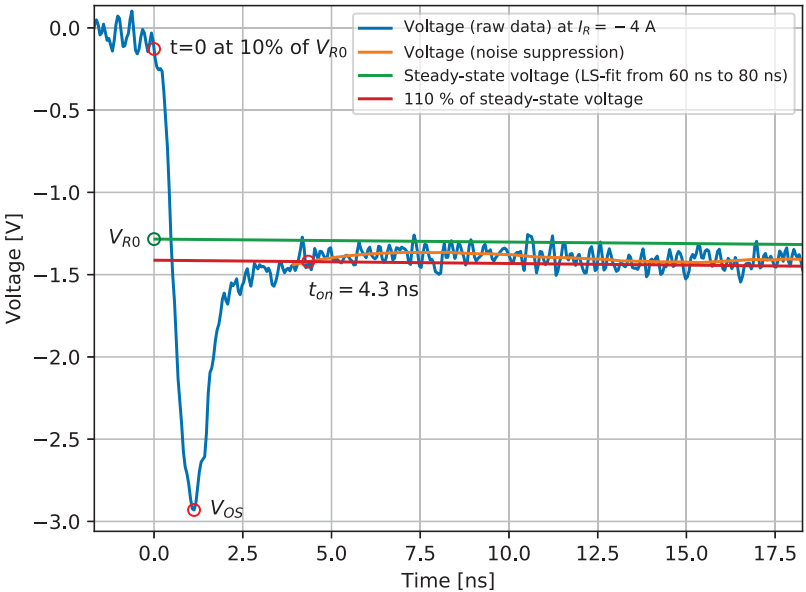


# D5V0M1U2LP3 [4], $I_R = -4$ A at 1 ns Rise Time

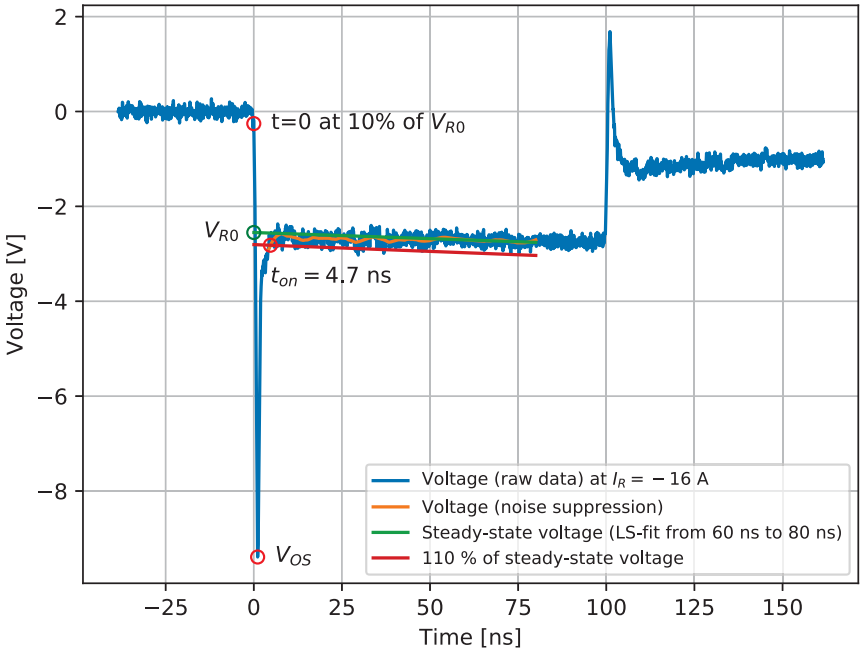


# D5V0M1U2LP3 [4], $I_R = -4$ A at 1 ns Rise Time

## Detail View

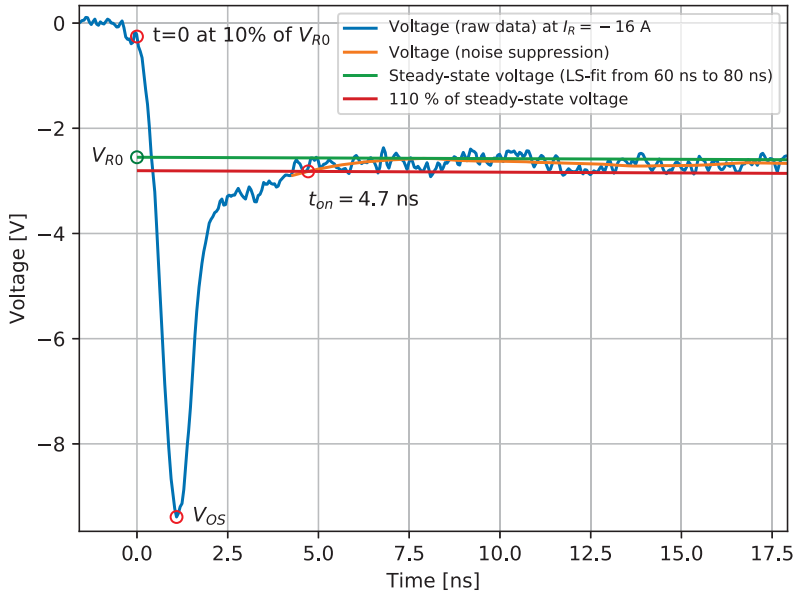


# D5V0M1U2LP3 [4], $I_R = -16$ A at 1 ns Rise Time



# D5V0M1U2LP3 [4], $I_R = -16$ A at 1 ns Rise Time

## Detail View



# Summary of Measurement Results

- Rreverse turn-on (overshoot recovery) time $t_{on}$ measurement results:

| | Reverse Mode | | | | Forward Mode | | | |
|---|---|---|---|---|---|---|---|---|
| | Rise Time = 100 ps | | Rise Time = 1 ns | | Rise Time = 100 ps | | Rise Time = 1 ns | |
| | IR = 4 A | IR = 16 A | IR = 4A | IR = 16 A | IR = -4 A | IR = -16 A | IR = -4A | IR = -16 A |
| **D5V0F1U2LP3** | 1.5 ns | 1.8 ns | 2.2 ns | 3.3 ns | 24.4 ns | 17.1 ns | 26.8 ns | 17.6 ns |
| **D5V0M1U2LP3** | 0.5 ns | 2.8 ns *) | 1.3 ns | 1.7 ns | 4 ns *) | 2 ns *) | 4.3 ns | 4.7 ns |

*) strong ringing

# Python script for $t_{on}$ extraction

## Experimental code download: https://www.hppi.de/files/extract_ton_2.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 06:59:36 2021

@author: Werner Simbuerger, HPPI
Python Code and Results:
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

"""

import os
import fnmatch
import zipfile
import io
import math
import requests
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from scipy import signal
import progressbar  # install progressbar2 package (not progessbar)
import pyperclip
from pathlib import Path


################################################################################
###  Parameter Definition ######################################################
################################################################################

i_extract = np.array([4, 16]) # extract the turn on time at these TLP currents
#d_path = Path(r"E:\PROJECT\TLP_3010C_R01\000_TLP_Data\2021_01_28_Diodes\01_D5V0P1B2LP3\100ps_neg_100ns")
d_path = Path(pyperclip.paste())

# position of the averaging window
avg_win_lo = 60 # averaging window start in [ns]
avg_win_hi = 80 # averaging window stop in [ns]

# turn-on time extraction threshold value in [percent] of the average clamping voltage
ton_threshold = 120

# extraction of steady state voltage:
# slope_mode = 'slope' ... calculate least square fit line in averaging window
# slope_mode = 'mean' ... calculate horizontal line at mean value in averaging window
slope_mode = 'mean'

# Noise suppression:
# https://plot.ly/python/smoothing/
# arg 1 window size used for filtering
# arg 2 order of fitted polynomial
#   optional: calculate fitting windows size from sampling rate
#       0.4 ns -> 5 GS/s -> window_size = 51
#       window_size = int(round_up_to_odd(1/delta_t*20.4))
```

```
    # take care about window size and filter order - good ranges are:
    # window size (always odd number): 51, 101, 201
    # filter order: 3, 5, 7
    window_size = 51
    filter_order = 3


    ##################################################################################################
    ###  Code Begin ##################################################################################
    ##################################################################################################

    def round_up_to_odd(f):
        return np.ceil(f) // 2 * 2 + 1

    # https://stackoverflow.com/questions/303200/how-do-i-remove-delete-a-folder-that-is-not-empty
    def remove_path(path: Path):
        if path.is_file() or path.is_symlink():
            path.unlink()
            return
        for p in path.iterdir():
            remove_path(p)
        path.rmdir()



    def list_files(folder='.', pattern='*', case_sensitive=False, subfolders=False):
        """Return a list of the file paths matching the pattern in the specified
        folder, optionally including files inside subfolders.
        """
        match = fnmatch.fnmatchcase if case_sensitive else fnmatch.fnmatch
        walked = os.walk(folder) if subfolders else [next(os.walk(folder))]
        return [os.path.join(root, f)
                for root, dirnames, filenames in walked
                for f in filenames if match(f, pattern)]


    def download_extract_zip(url):
        """
        Download a ZIP file and extract its contents in memory
        yields (filename, file-like object) pairs
        """
        response = requests.get(url)
        with zipfile.ZipFile(io.BytesIO(response.content)) as thezip:
            for zipinfo in thezip.infolist():
                with thezip.open(zipinfo) as thefile:
                    yield zipinfo.filename, thefile

    def file_extract_zip(file_name):
        """
        Read ZIP file and extract its contents in memory
        yields (filename, file-like object) pairs
        """
        with open(file_name,'rb') as f:
            file_content = f.read()

    #    response = requests.get(url)
        with zipfile.ZipFile(io.BytesIO(file_content)) as thezip:
            for zipinfo in thezip.infolist():
                with thezip.open(zipinfo) as thefile:
                    yield zipinfo.filename, thefile


    def find_nearest(array, values):    #https://stackoverflow.com/questions/2566412/find-nearest-value-in-numpy-array
        array = np.asarray(array)
```

```
        # the last dim must be 1 to broadcast in (array - values) below.
        values = np.expand_dims(values, axis=-1)
        indices = np.abs(array - values).argmin(axis=-1)
        return indices
    #    return array[indices]


    def fitxy(x,y):
        X_b = np.c_[np.ones((np.size(x), 1)), x]  # add x0 = 1 to each instance
        theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
        d = theta[0]
        k = theta[1]
        return(k*x+d)

    def fitxy_kd(x,y):
        X_b = np.c_[np.ones((np.size(x), 1)), x]  # add x0 = 1 to each instance
        theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
        d = theta[0]
        k = theta[1]
        return(k,d)


    def TriggerEdges(y,trigger_val):    #https://stackoverflow.com/questions/50365310/python-rising-falling-edge-oscilloscope-like-trigger
        mask1 = (y[:-1] <= trigger_val) & (y[1:] >= trigger_val)
        mask2 = (y[:-1] >= trigger_val) & (y[1:] <= trigger_val)
        return(np.flatnonzero(mask1 | mask2)+1) # returns index of rising and falling edge


    def yoffset(x,y):
        ymax = max(y)
        ymin = min(y)
        x1 = x[0]
        if abs(ymax) > abs(ymin):  # positive HBM Pulse
            x2 = x[TriggerEdges(y/abs(ymax),0.5)[0]]
        else: # negative HBM Pulse
            x2 = x[TriggerEdges(y/abs(ymin),-0.5)[0]]
        y0 = np.mean(y[np.where(x < (0.7*(x2-x1) + x1))])
        #print('\n\r'+'ymin='+str(ymin)+','+'ymax='+str(ymax)+','+'x1='+str(x1)+','+'x2='+str(x2)+','+'y0='+str(y0))
        return(y0)


    def TimeStamp(): # create 10-digit time stamp number (every second increment)
        dt = datetime.now()
        mdn = dt + timedelta(days = 366)
        frac = (dt-datetime(dt.year,dt.month,dt.day,0,0,0)).seconds / (24.0 * 60.0 * 60.0)
        day = mdn.toordinal() + frac
        return(math.ceil((day-7e5)*1e5))

    def interpolated_intercepts(x, y1, y2): #https://stackoverflow.com/questions/42464334/find-the-intersection-of-two-curves-given-by-x-y-data-with-high-precision-in
        """Find the intercepts of two curves, given by the same x data"""

        def intercept(point1, point2, point3, point4):
            """find the intersection between two lines
            the first line is defined by the line between point1 and point2
            the first line is defined by the line between point3 and point4
            each point is an (x,y) tuple.

            So, for example, you can find the intersection between
            intercept((0,0), (1,1), (0,1), (1,0)) = (0.5, 0.5)

            Returns: the intercept, in (x,y) format
            """
```

```
            def line(p1, p2):
                A = (p1[1] - p2[1])
                B = (p2[0] - p1[0])
                C = (p1[0]*p2[1] - p2[0]*p1[1])
                return A, B, -C

            def intersection(L1, L2):
                D  = L1[0] * L2[1] - L1[1] * L2[0]
                Dx = L1[2] * L2[1] - L1[1] * L2[2]
                Dy = L1[0] * L2[2] - L1[2] * L2[0]

                x = Dx / D
                y = Dy / D
                return x,y

            L1 = line([point1[0],point1[1]], [point2[0],point2[1]])
            L2 = line([point3[0],point3[1]], [point4[0],point4[1]])

            R = intersection(L1, L2)

            return R

        idxs = np.argwhere(np.diff(np.sign(y1 - y2)) != 0)

        xcs = []
        ycs = []

        for idx in idxs:
            xc, yc = intercept((x[idx], y1[idx]),((x[idx+1], y1[idx+1])), ((x[idx], y2[idx])), ((x[idx+1], y2[idx+1])))
            xcs.append(xc)
            ycs.append(yc)
        return np.array(xcs), np.array(ycs)


    # Werner:  START
    if __name__ == "__main__":

        # r_path = d_path.joinpath('report')

        col = 1; # column in data file:  0...t, 1...v(t), 2...i(t)
        # if Path(r_path).exists():
        #     remove_path(r_path) # force folder delete
        # os.mkdir(r_path)    # create temporary report folder


        d_files = list_files(folder=d_path, pattern='*.zip', case_sensitive=False, subfolders=True)

        plt.close("all")

        response = file_extract_zip(d_files[0])
        data_dict = {}
        pulse_voltage_text_list = []; # pulse voltage text list
        iv_data = []; # iv data
        pulse_voltage_array = []; # pulse voltage array
        for f in response:
            iv_dat = np.genfromtxt(f[1], delimiter=',',skip_header=1)
            data_dict[f[0]] = iv_dat
            pulse_voltage_text_list.append(f[0])
            iv_data.append(iv_dat)

        p_list = list(data_dict)
        for i in pulse_voltage_text_list:
            pulse_voltage_array.append(i[:-5].split('_')[1]) # create pulse voltage list
```

```
244         pulse_voltage_array = np.array([float(i) for i in pulse_voltage_array]) #create pulse voltage array
245
246
247         skip_rows = 0
248         with open(d_path.joinpath('TLP_data.csv')) as fp:
249             for line in fp:
250                 skip_rows += 1
251                 if "Index" in line:
252                     break
253
254         TLP_data = np.loadtxt(d_path.joinpath('TLP_data.csv'), delimiter=',', skiprows=skip_rows)
255
256         iv = find_nearest(abs(TLP_data[:,3]), abs(i_extract))
257
258
259         bar = progressbar.ProgressBar(max_value=np.size(iv))
260         bar_cnt = 1
261
262
263
264
265         for m in iv:
266
267             bar.update(bar_cnt)
268             bar_cnt += 1
269
270             x = iv_data[m][:,0]
271             delta_t = (x[1]-x[0]) # calculate osc sampling rate
272             y = iv_data[m][:,col]  # 0...t, 1...V(t), 2...I Intern, 3... I CT-2(t) -> Default (2)
273             y_i = iv_data[m][:,2] # Strom Transiente
274             v_p = pulse_voltage_array[m];
275
276
277
278             y0 = yoffset(x,y)
279             y = y - y0
280
281             ymax = max(abs(y))
282             i = np.where(abs(y) == ymax)
283             tmax = x[i]
284             ymax = np.sign(y[i])*ymax
285             i = np.where(x <= tmax)
286             xr = x[i]
287             yr = y[i]
288             xds, yds = interpolated_intercepts(xr,yr,xr*0.0+0.1*ymax)
289             x = x - xds[0] ##############################################################################################################
290             xr = xr - xds[0]
291
292
293             i = np.where((x >= avg_win_lo) & (x <= avg_win_hi))
294             x1 = x[i]
295             y1 = y[i]
296
297
298             if slope_mode == 'slope':
299                 k,d = fitxy_kd(x1,y1)
300             if slope_mode == 'mean':
301                 k = 0
302                 d = np.mean(y1)
303
304             vr0 = d
305             xds, yds = interpolated_intercepts(xr,yr,xr*0.0+0.1*vr0)
306             x = x - xds[-1] ##############################################################################################################
307             xr = xr - xds[-1]
```

```
308
309             i = np.where((x >= 0) & (x <= avg_win_hi))
310             x1 = x[i]
311             y1 = k*x[i]+d
312             x1spec = x[i]
313             y1spec = y1 * ton_threshold/100.0
314
315             ymax = max(abs(y))
316             i = np.where(abs(y) == ymax)
317             tmax = x[i]
318             ymax = np.sign(y[i])*ymax
319
320             i = np.where((x >= tmax) & (x <= avg_win_hi)) ########################################################
321             xns = x[i]
322             yns = y[i]
323             # Noise suppression:
324             # https://plot.ly/python/smoothing/
325             # arg 1 window size used for filtering
326             # arg 2 order of fitted polynomial
327             #   optional: calculate fitting windows size from sampling rate
328             #       0.4 ns -> 5 GS/s -> window_size = 51
329             #       window_size = int(round_up_to_odd(1/delta_t*20.4))
330             # take care about window size and filter order - good ranges are:
331             # window size (always odd number): 51, 101, 201
332             # filter order: 3, 5, 7
333             ys=signal.savgol_filter(yns, window_size, filter_order)
334
335
336             ton, yon = interpolated_intercepts(xns,ys,(k*xns+d)*float(ton_threshold/100))
337             ton = ton[0] ############################################################
338             yon = yon[0] ############################################################
339             i = np.where(xns >= ton[0]*0.9)
340             xns1 = xns[i]
341             ys1 = ys[i]
342
343             i = np.where((x >= avg_win_lo) & (x <= avg_win_hi))
344             y_im = y_i[i]
345             i_mean = np.mean(y_im)
346
347
348
349             fig, ax = plt.subplots()
350             ax.plot(x,y,label=("Voltage (raw data) at $I_R = %.0f$ A" % i_mean))
351
352             ax.plot(xns1,ys1,label='Voltage (noise suppression)')
353             if slope_mode == 'slope':
354                 ax.plot(x1,y1,label=("Steady-state voltage (LS-fit from %g ns to %g ns)" % (avg_win_lo, avg_win_hi)))
355             if slope_mode == 'mean':
356                 ax.plot(x1,y1,label=("Steady-state voltage (mean at %g ns to %g ns)" % (avg_win_lo, avg_win_hi)))
357
358
359             ax.plot(x1spec,y1spec,label=("%g %% of steady-state voltage" % float(ton_threshold)))
360
361             ax.plot(0,0.1*vr0,'o',color='r',mfc='none')
362             ax.annotate("t=0 at 10% of $V_{R0}$", xy=(0,0.1*vr0),xycoords='data',xytext=(8, 0), textcoords='offset points',ha='left',va="center")
363
364             ax.plot(tmax,ymax,'o',color='r',mfc='none')
365             ax.annotate(("$V_{OS}=%.1f$ V" % ymax), xy=(tmax,ymax),xycoords='data',xytext=(10, 0), textcoords='offset points',ha='left',va="center")
366
367
368             ax.plot(0,vr0,'o',color='g',mfc='none')
369             ax.annotate(("$V_{R0}=%.1f$ V" % vr0), xy=(0, vr0),xycoords='data',xytext=(-70, 0), textcoords='offset points',ha='left',va="center")
370
371             ax.plot(ton,yon,'o',color='r',mfc='none')
```

```
372         ax.annotate(("$t_{on}=%.1f$ ns" % ton), xy=(ton,yon),xycoords='data',xytext=(0, 15*np.sign(yon)), textcoords='offset points',ha='left',va="center")
373
374
375         ax.set(xlabel='Time [ns]', ylabel='Voltage [V]')
376         ax.grid()
377         ax.legend(loc='best',fontsize=8)
378
379         devfname = r"%g" % (v_p, )
380         devfname = devfname.replace(".","_")
381
382         # pltfname = r_path.joinpath(d_path.parts[-2] + "_" + devfname + ".pdf")
383         # plt.savefig(pltfname, bbox_inches='tight')
384
385
386
387         # pyperclip.copy(txt)
```

# References


[1] Vishay General Semiconductor. (Aug. 11, 2011), "Physical Explanation," Application Note, [Online]. Available: https://www.vishay.com/docs/84064/anphyexp.pdf (visited on 01/19/2019).

[2] HPPI (High Power Pulse Instruments) GmbH. (Jan. 19, 2019), "Charge Recovery Measurements with TLP," Application Note, [Online]. Available: https://www.hppi.de/files/Charge_Recovery_Measurements_with_TLP.pdf (visited on 01/19/2019).

[3] Diodes Incorporated. (2019), "D5V0F1U2LP3," [Online]. Available: https://www.diodes.com/products/discrete/protection-devices/data-line-protection/part/D5V0F1U2LP3 (visited on 02/02/2019).

[4] ——, (2019), "D5V0M1U2LP3," [Online]. Available: https://www.diodes.com/products/discrete/protection-devices/data-line-protection/part/D5V0F1U2LP3 (visited on 02/02/2019).