Status report - Thursday, April 23nd

Threshold Scans and Data Evaluation

Maurice Donner

23. April 2020

How things have been done

General Process - Flow Diagram

Perform a set of measurements in lab (Python) $\rightarrow$

How things have been done

General Process - Flow Diagram

Perform a set of measurements in lab (Python) $\rightarrow$ $\rightarrow$ Plot results (Python)

How things have been done

General Process - Flow Diagram

$$\begin{tikzcd}
\text{Perform a set of measurements in lab (Python)} \arrow[r] & \text{Write results to .csv (done by hand)} \arrow[r] & \text{Plot results (Python)}
\end{tikzcd}$$

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file
- Extract and analyze measurement data

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file
- Extract and analyze measurement data
- Write that information into a csv file for further analysis

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file
- Extract and analyze measurement data
- Write that information into a csv file for further analysis

Approach

Simplyfying the process by writing a script that

- Extracts the **timestamp** for each measurement and properly relates the .cfg to the .dat files

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file
- Extract and analyze measurement data
- Write that information into a csv file for further analysis

Approach

Simplyfying the process by writing a script that

- Extracts the **timestamp** for each measurement and properly relates the .cfg to the .dat files
- performes analysis on ALL of the measurement data at once

Working on a solution

Problem: Usually quite large set of data

For each measurement we have to

- Extract Parameters used (VCASN, ITHR) from the specific config file
- Extract and analyze measurement data
- Write that information into a csv file for further analysis

Approach

Simplyfying the process by writing a script that

- Extracts the **timestamp** for each measurement and properly relates the .cfg to the .dat files
- performes analysis on ALL of the measurement data at once
- writes result to a csv file

Scripting in Bash

ScanConfig_**200121_193551**.cfg
ScanConfig_**200121_193951**.cfg
ThresholdScan_**200121_193551**.dat
ThresholdScan_**200121_193951**.dat

Scripting in Bash

ScanConfig_**200121_193551**.cfg
ScanConfig_**200121_193951**.cfg
ThresholdScan_**200121_193551**.dat
ThresholdScan_**200121_193951**.dat

```
for i in $(ls $PATHTOFILES | grep '.dat'); do
#Extract Timestamp
TIMESTAMP=$(echo $i | tail -c 18 | head -c 13)
CONFIG="ScanConfig_$TIMESTAMP.cfg"

#Then extract Parameters from config file (Later add VBB)
VCASN=$(cat $PATHTOFILES$CONFIG | grep 'VCASN' | awk -F ' ' '{print $2}' | head -1)
ITHR=$(cat $PATHTOFILES$CONFIG | grep 'ITHR' | awk -F ' ' '{print $2}')

TRSH=$(./thresh.py $PATHTOFILES$i)

# Write to csv file
printf '%s\n' "$TIMESTAMP" "$VCASN" "$ITHR" "$TRSH" | paste -sd ',' >> output.csv
done
```

Scripting in Bash

ScanConfig_**200121_193551**.cfg
ScanConfig_**200121_193951**.cfg
ThresholdScan_**200121_193551**.dat
ThresholdScan_**200121_193951**.dat

```
for i in $(ls $PATHTOFILES | grep '.dat'); do
#Extract Timestamp
TIMESTAMP=$(echo $i | tail -c 18 | head -c 13)
CONFIG="ScanConfig_$TIMESTAMP.cfg"

#Then extract Parameters from config file (Later add VBB)
VCASN=$(cat $PATHTOFILES$CONFIG | grep 'VCASN' | awk -F ' ' '{print $2}' | head -1)
ITHR=$(cat $PATHTOFILES$CONFIG | grep 'ITHR' | awk -F ' ' '{print $2}')

TRSH=$(./thresh.py $PATHTOFILES$i)

# Write to csv file
printf '%s\n' "$TIMESTAMP" "$VCASN" "$ITHR" "$TRSH" | paste -sd ',' >> output.csv
done
```

```
Timestamp,VCASN,ITHR,Threshold [DAC]
200121_193551,47,51,13.60355155825141
200121_193951,47,60,15.953068558715911
...
```

Plotting

New Problem

Data is not ordered, and the csv contains multiple entries for the same values of VCASN and ITHR

Plotting

New Problem

Data is not ordered, and the csv contains multiple entries for the same values of VCASN and ITHR

$\rightarrow$ When doing plots, formerly used hardcoding

Plotting

New Problem

Data is not ordered, and the csv contains multiple entries for the same values of VCASN and ITHR
→ When doing plots, formerly used hardcoding

Approach

Write a ”sorting” algorithm, that automatically identifies the ranges chosen for VCASN and ITHR.

```
Timestamp,VCASN,ITHR,Threshold [DAC]
200323_134855,53,51,6.725833998523066
200323_135218,53,60,8.186236372633545
200323_135541,53,70,9.833234746846962
200323_132837,50,51,9.335578330893117
200323_133200,50,60,11.202496464178642
200323_133523,50,70,13.211766207119839
```

→

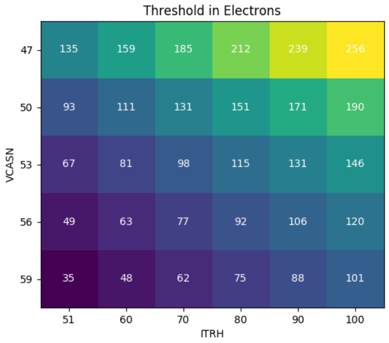
```
VCASN = array([50,53])
ITHR = array([51,60,70])
```

plotting

```
VCASN, ITHR, TRSH = np.loadtxt(csv, skiprows=1, usecols=(1,2,3), delimiter=",", unpack=True)

def getValues(array):
    #Create a temporary list
    temp = []
    #Write each unique entry into the temporary list
    for i in array:
        if i in temp: continue
        else: temp.append(i)
    #Since the array of values in this case is quite small, we can use temp.sort
    temp.sort()
    output = np.ndarray((len(temp)),dtype=int)
    for i in range(len(temp)):
        output[i] = int(temp[i])
    return output

VCASN_0 = getValues(VCASN)
ITHR_0 = getValues(ITHR)

##### Implement sorting algorithm #####
Threshold = np.ndarray((len(VCASN_0),len(ITHR_0)))
for i in range(len(VCASN_0)):
    for j in range(len(ITHR_0)):
        Threshold[i,j] = TRSH[(VCASN == VCASN_0[i]) & (ITHR == ITHR_0[j])]
#######################################
```

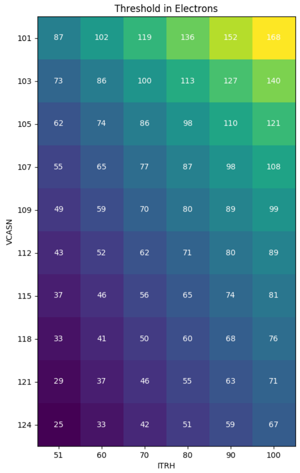
Results

0 V Back Bias



VCASN \ ITRH	51	60	70	80	90	100
47	135	159	185	212	239	256
50	93	111	131	151	171	190
53	67	81	98	115	131	146
56	49	63	77	92	106	120
59	35	48	62	75	88	101

3 V Back Bias



VCASN \ ITRH	51	60	70	80	90	100
101	87	102	119	136	152	168
103	73	86	100	113	127	140
105	62	74	86	98	110	121
107	55	65	77	87	98	108
109	49	59	70	80	89	99
112	43	52	62	71	80	89
115	37	46	56	65	74	81
118	33	41	50	60	68	76
121	29	37	46	55	63	71
124	25	33	42	51	59	67

Note: This is a corrected Version of the presentation. The Errors in data have been identified and fixed. The missing config file has been restored, and faulty values have been masked entirely. This is an accurate representation of all values.

For the future

Let the bash script...

- Include Errors

For the future

Let the bash script...

- Include Errors
- Automatically detect faulty runs

For the future

Let the bash script...

- Include Errors
- Automatically detect faulty runs
- Work on different kinds of scans (NOISEOCC etc.)

For the future

Let the bash script...

- Include Errors
- Automatically detect faulty runs
- Work on different kinds of scans (NOISEOCC etc.)

Thank you!