

# **Investigation of a Low-Cost Heterodyne Interferometry System for Diagnostics for MAST-U**

Summer Placement Project 2017

Christopher J. Hickling

August 25, 2017

# Contents

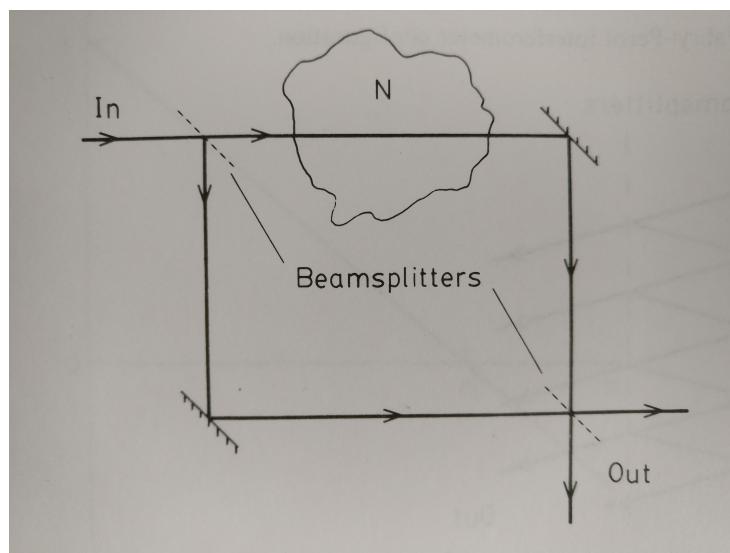
<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>2</b>
2.1	Interferometry . . . . .	2
<b>3</b>	<b>Experimental Set-up</b>	<b>4</b>
3.1	Red Pitaya . . . . .	4
3.2	Koheron Laser Board . . . . .	5
3.3	Koheron Client . . . . .	7
3.4	Double Mach-Zehnder Interferometer Setup . . . . .	8
<b>4</b>	<b>Experimental</b>	<b>9</b>
4.1	Collecting data . . . . .	9
4.2	Manipulating Data . . . . .	10
4.3	Inducing a phase change by the use of a speaker . . . . .	16
<b>5</b>	<b>Bibliography</b>	<b>18</b>
<b>Appendix A</b>	<b>Code</b>	<b>ii</b>
A.1	Saving Data . . . . .	ii
A.2	Hilbert Transform on Saved Data . . . . .	v
A.3	Hilbert Transform in Real-time . . . . .	x

# Abstract

The aim of this project is to investigate and test the effectiveness of a new interferometer system for diagnostic purposes. This system is based on the use of a 1550nm laser diode interferometer system designed to measure the electron density of plasma in MAST-U (Mega Amp Spherical Tokamak - Upgrade). The benefit of using this new laser system over the old is that because of the low cost, and lack of health risks, many interferometer channels can be made to look at multiple interesting points of the Tokamak, e.g. X-Point in divertor region.

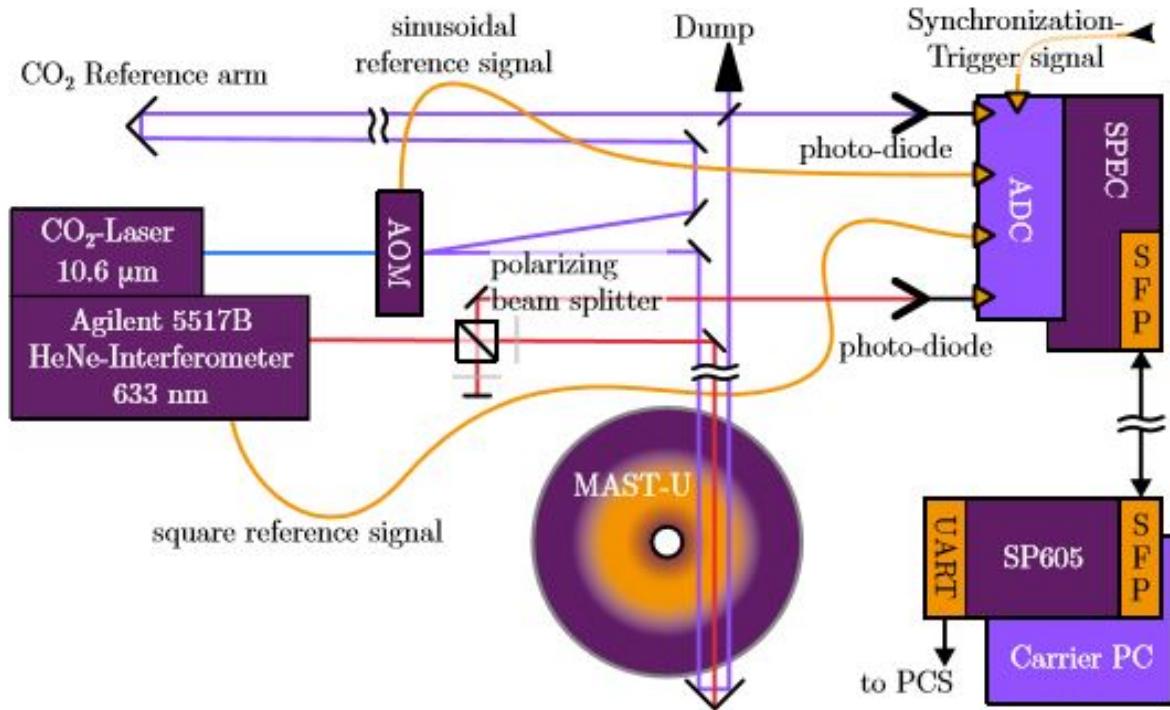
# Introduction

## 2.1 Interferometry



**Figure 2.1.** A schematic for a Mach-Zehnder interferometer where the Region N corresponds to a region of plasma. Image taken from Principles of Plasma Diagnostics [1, p. 97]

An interferometer, by design, is an incredibly sensitive piece of distance measuring apparatus. They work on the principle of interference between two laser beam paths. If a laser is split into two beam paths of unequal lengths, an interference pattern will form if they are recombined. The rate at which this interference pattern chirps is proportional to the difference in path length of the arms. If, for example, the path lengths were identical, no interference would take place upon recombination. By inserting a plasma into one of these arms a path difference will become apparent as the density will change the refractive index of the plasma. A higher density results in a higher refractive index, and therefore a larger apparent path length. Usually a standard Mach-Zehnder interferometer set-up is used [1], seen in figure 2.1, as this uses two arms that do not pass each other.



**Figure 2.2.** A schematic of the interferometer currently in place on MAST-U. This shows the complexity of the system and the difficulty of expanding it to include more interferometers. [2]

Currently in MAST (Mega Amp Spherical Tokamak) a heterodyne system is in place. This consists of a  $CO_2$  laser in combination with a HeNe laser integrated with a FPGA system designed by J.K.Brunner [2]. The choice behind these lasers is that  $10.6\mu m$  wavelength has strong interaction with the plasma. The HeNe does not have a high level of interaction with the plasma. This means the two lasers can be sent along the same beam path and interfere. So long as the modulation frequency of each of the lasers is known to a high precision, the plasma density can then be derived. One of the largest limitations to this system is the temperature dependence of the  $CO_2$  laser. The laser needs to be water-cooled in order to have stability in temperature. If it does not have a stable temperature the laser output will be effected, and since this system by design is sensitive, this would heavily effect results.

Another limitation is the cost of the  $CO_2$  laser, as it is very expensive and requires a large amount of maintenance. The wavelength of the laser is chosen to have strong interaction with the plasma, but it also has strong interaction with human skin and eyes. As a result this laser has very strict safety precautions and requirements that come with its installation. These precautions are made more apparent since the laser has to enter the vacuum vessel via a glass viewing port which leaves the beam path exposed for a short distance.

# Experimental Set-up

## 3.1 Red Pitaya

The Red Pitaya (RP) [3] is a open source micro controller with a built in Field Programmable Gate Array (FPGA) with a very user friendly user interface. It operates two inputs and two outputs at 125MS/s. FPGA's are notorious for being difficult to program as they are intrinsically complicated. The stand out function of a FPGA is the way it processes information. It can perform many tasks in parallel pipelined processing i.e. performing the same operation on many data sets. The number of parallel pipelines that can be implemented depends entirely on the hardware limitations of the FPGA. This often means that the limiting factor of the speed at which a RP can process data is the speed at which you can transmit data to it. This gives FPGA's a large advantage over conventional computers when it comes to real time data applications.



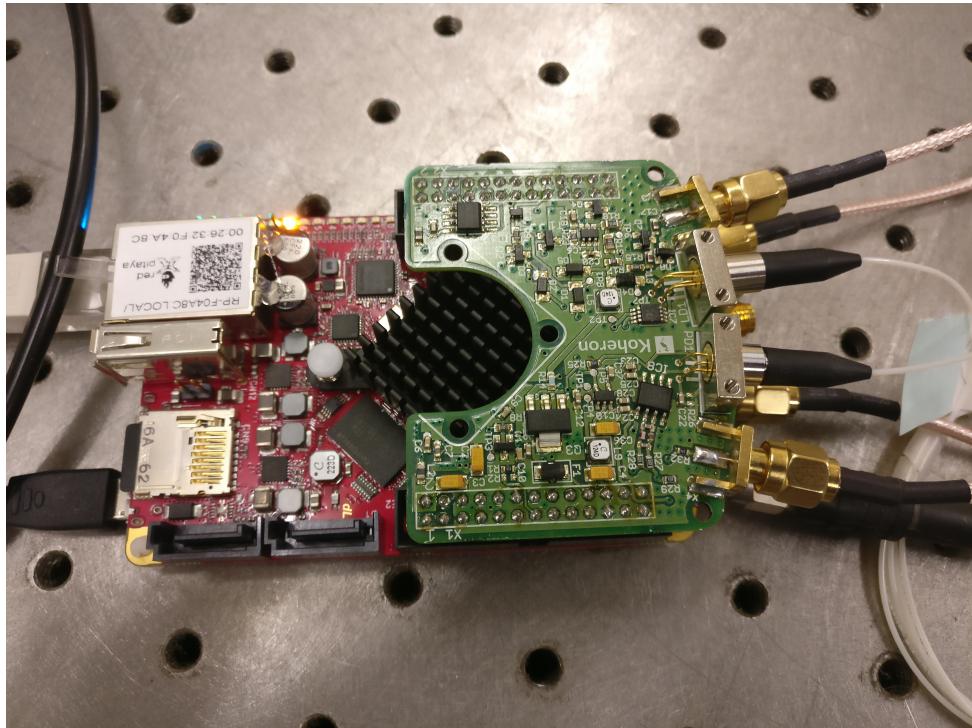
**Figure 3.1.** The Red Pitaya showing all the connected ports. Each of the analogue ports can sample at 125MHz.

Name	Type	Connector	Description
IN1/IN2	Input	SMA-F	RF input(High-Z, $1M\Omega$ // $10pF$ )
OUT1 / OUT2	Output	SMA-F	RF output ( $50\Omega$ )
Ethernet	Full Duplex	RJ45	1000 Base-T Ethernet Connection
USB	Full Duplex	A USB	Used for standard USB Devices
Micro USB (Console)	Full Duplex	Micro B USB	Used for console connection
Micro USB (Power)	Input	Micro B USB	5V / 2A power supply
Micro SD	Full Duplex	Micro SD slot	Micro SD memory card

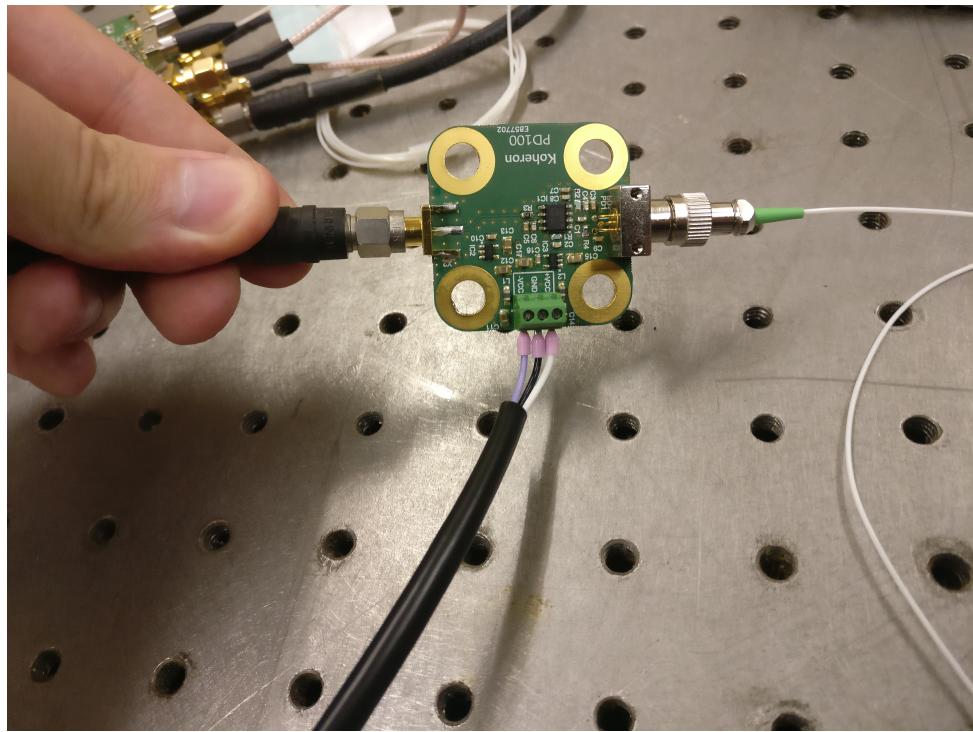
**Table 3.1.** Full Specifications of the Red Pitaya's input/output ports. Both fast analogue inputs and fast analogue outputs operate at 125MHz per channel.

## 3.2 Koheron Laser Board

The Koheron Laser board [4] that docks onto the RP is equipped with a 1550nm laser and a 60MHz photo-detector, this can be seen in figure 3.2. The laser has a maximum output of 3mW which marks the laser as a Class 1, meaning it is safe for any activity with no need for personal protective equipment. In addition to this to enable a second channel an addition of a Koheron PD100 photodetector board [5] (figure 3.3) is also needed. Using the RP and the Koheron Laser Board a simple interferometer can be built [6].



**Figure 3.2.** An image of the Red Pitaya with the laser board attached. From the top of the image to the bottom, the laser board has the connections; Input (setting laser current and modulation from the RP), laser output, laser input (photodetector) and output (connects to the RP).



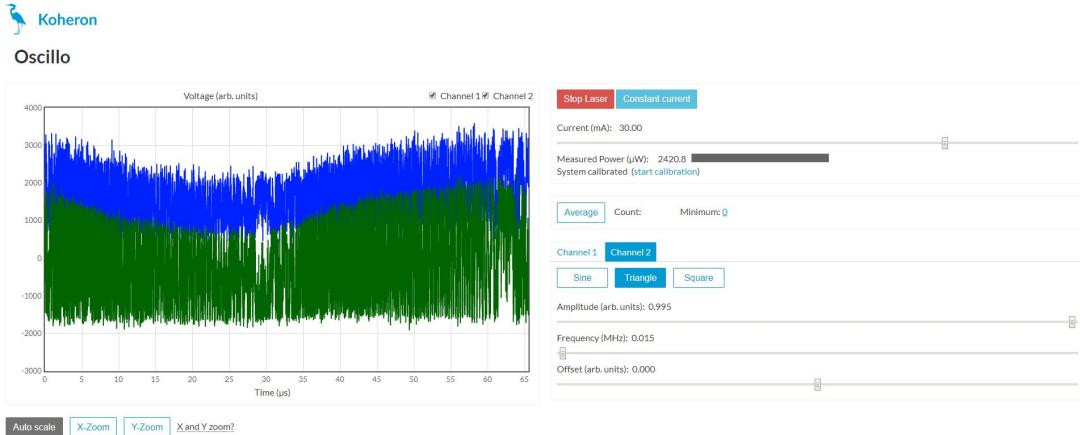
**Figure 3.3.** The PD100 laser board takes a fibre optic input and converts it into an electrical signal using an Indium Gallium Arsenide photo diode. The PD100 requires a DC current rated between -15 to +15V. Best optimization was found at -6.8V and +6.8V. Increasing this voltage causes the signal to saturate.

### 3.3 Koheron Client

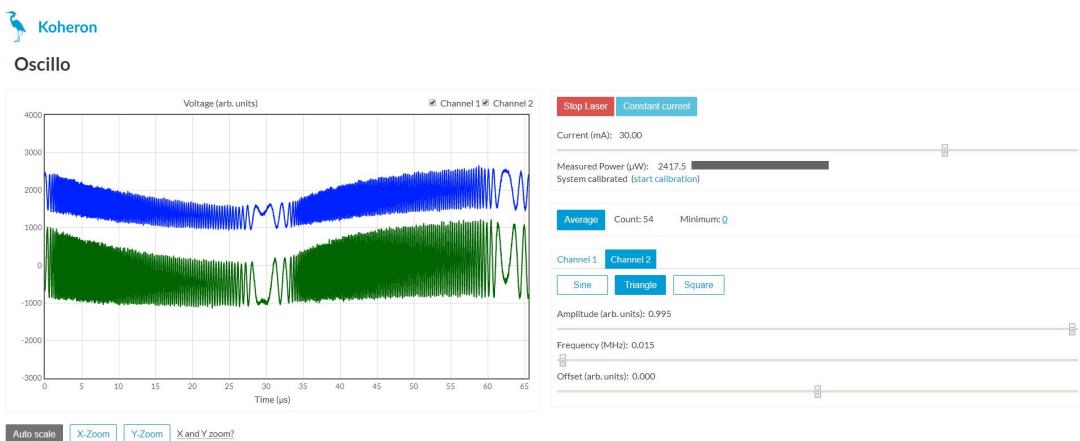
As part of the received package from Koheron, the RP comes preloaded with the software to run the Koheron client [7]. This gives full control over the laser system including a visual output of the data collected by the two inputs on the board. The main benefit of this is that it gives an immediate representation of what applying different modulation frequencies to the laser results in. The number of chirps that can be seen in a discreet period of  $65\mu s$  is dependant on the modulation frequency applied. The chirps always occur at the maxima or minima of the modulation. In an entire wavelength there should be one maxima and one minima. Modulation can also only be applied in integer divisions between 4096 (equation 3.2) and 8192 (equation 3.1). This is a limitation designated by the hardware. Hence, modulations can be achieved between the following values.

$$125 * 10^6 Hz / 8192 = 15300 Hz = 15.3 KHz \quad (3.1)$$

$$125 * 10^6 Hz / 4096 = 62500000 Hz = 62.5 MHz \quad (3.2)$$



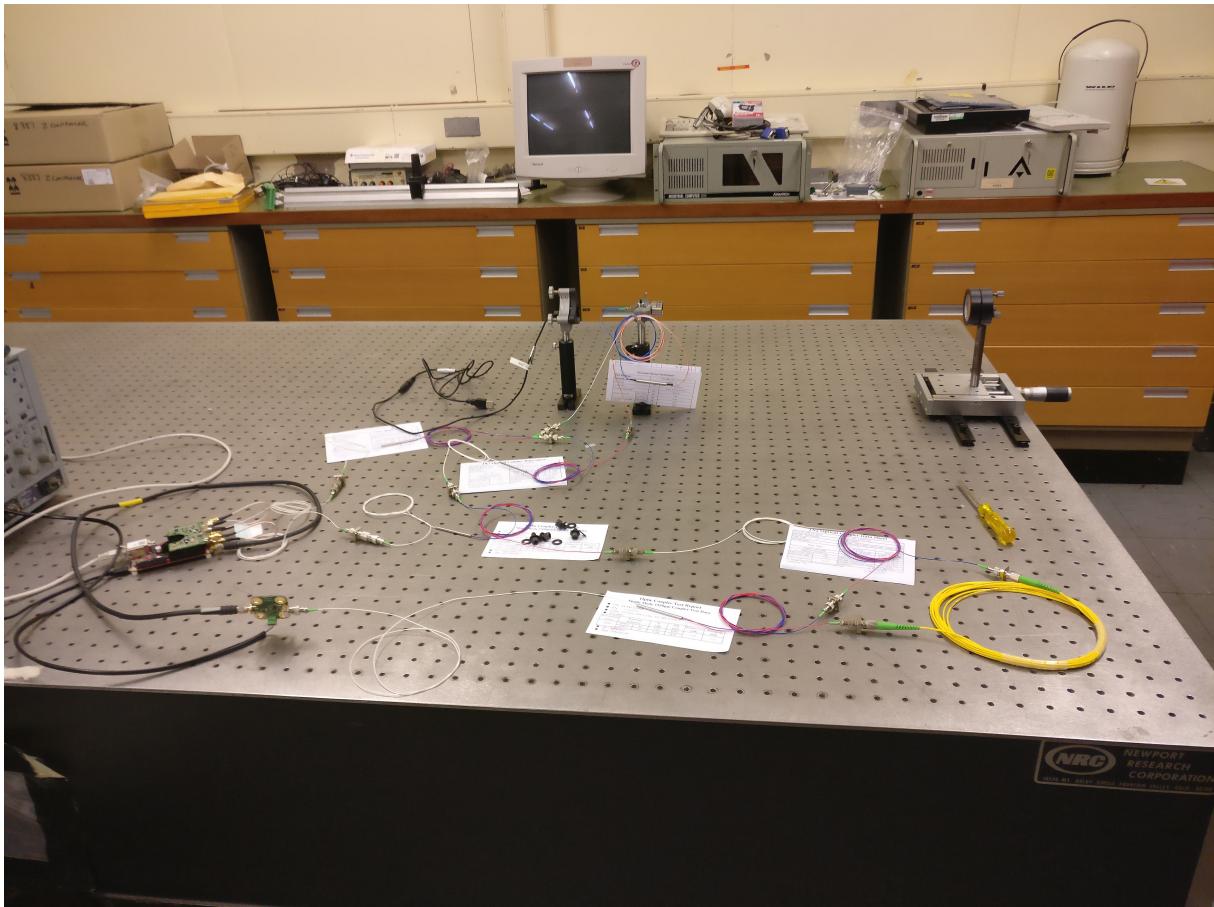
**Figure 3.4.** Raw data within the oscillo client. The way in which the current and modulation are set can be seen. The blue dataset shows the scene interferometer, the green shows the reference.



**Figure 3.5.** Averaged data within the oscillo client. The way in which the current and modulation are set can be seen. The blue dataset shows the scene interferometer, the green shows the reference.

### 3.4 Double Mach-Zehnder Interferometer Setup

By setting up two Mach-Zehnder interferometers using the same laser input but having isolated photodetector outputs a more realistic temperature correction can be made. The system is made out of single mode fibres, using couplers as beam splitters. This is because if you actually have two interferometers, a reference and a scene, each consisting of two arms. The reference interferometer has a beam splitter to split the beam, a 5 meter extension and then a beam combiner. This causes a consistent interference pattern where the rate of chirp is proportional to the temperature difference within the laser diode.



**Figure 3.6.** The full setup showing the two interferometers fully connected. The scene extension of the scene interferometer can be seen at the back, as the airgap is between the collimator and the translation stage. The reference interferometer extension is the yellow winding of single mode fibre cable in the bottom right. There is a laser diode behind the collimator only for the purpose of aligning the mirror.

This is in contrast to the scene interferometer, where the extension is provided by 4m of circulator fibres, into a collimator and a half a meter air gap to reflect off of a mirror and back into the collimator (1m air gap in total). This results in the same 5m extension but this interferometer has phase shifts proportional to the temperature fluctuations and phase differences caused by vibrations in the mirror.

# Experimental

## 4.1 Collecting data

In order to collect data from the RP code must be written to interface with the RP and the Koheron Client. There are already many inbuilt drivers and scripts in place but none of them fulfil all the needs of the project. Firstly in order to allow a computer to control the RP the Koheron package has to be installed. This was done via the software Anaconda [8] which contains python 2.7 [9] as well as many inbuilt python packages such as numpy [10] and scipy [11]. Using the Anaconda Prompt and typing "pip install koheron", the required packages will be downloaded to the computer. The closest program to the needs was "Oscillo.py". This had means to gather data from the DAC and also control the current supplied to the laser. By adding in the code and drivers required to define a modulation to the laser, the board could then be controlled entirely by a Python API instead of the client. Data can be collected from the DAC directly once the client has been interfaced with, this can be seen in appendix A.1.

Data collection is handled by decimated data, a driver written by Koheron (contained within the software development kit [7]). The way this driver works is it collects 16384 data points directly from the DAC and transfers them to the PC. In order to collect the data as quickly as possible, this data collection is just performed in a single loop with no processing done on it. The collection and post processing could be done simultaneously at the cost of resolution of data. Currently 500 data sets ( $8192^2$ ) can be collected in 5 seconds. This can be converted into a frequency by the following:

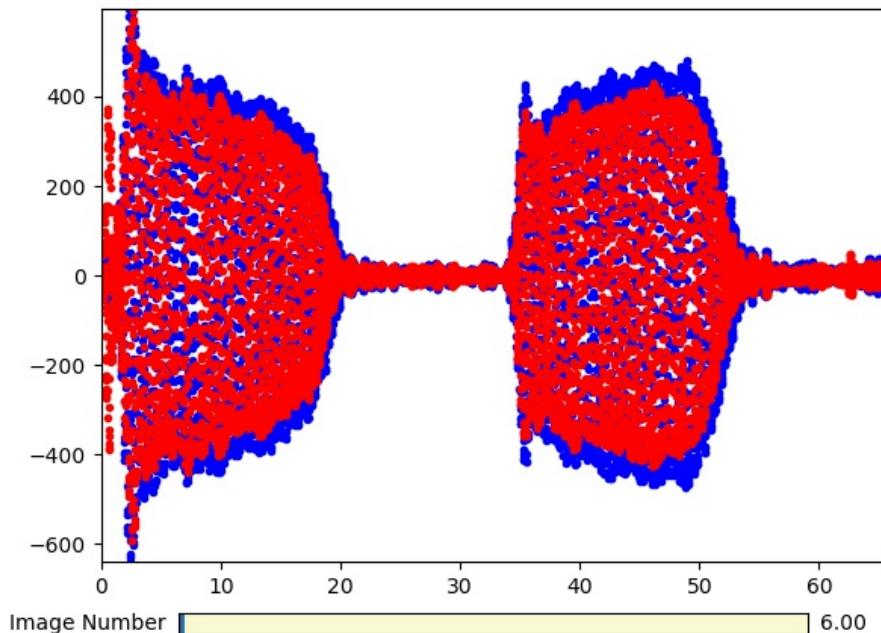
$$\begin{aligned} 8192 * 2 &= 16384 \\ (500 * 16384) / 5 &= 1638400 = 1.64 MHz \\ 1.64 / 125 &= 0.0131072 = 1.31\% \end{aligned} \tag{4.1}$$

This means that in its current form, as shown by equation 4.1, 98.7% of data is not transferred to the PC. This is mostly due to the fact that the RP can handle the data much faster than the PC can accept it. This results in the PC in the system being a huge bottleneck. A potential fix for this could be to set-up a second RP as a server that sits in between the PC and the first RP. This would save data directly into its RAM and pass the data to the PC at a rate the PC can handle. This would not be useful in a real-time application scenario, but in terms of data analysis this could massively increase resolution.

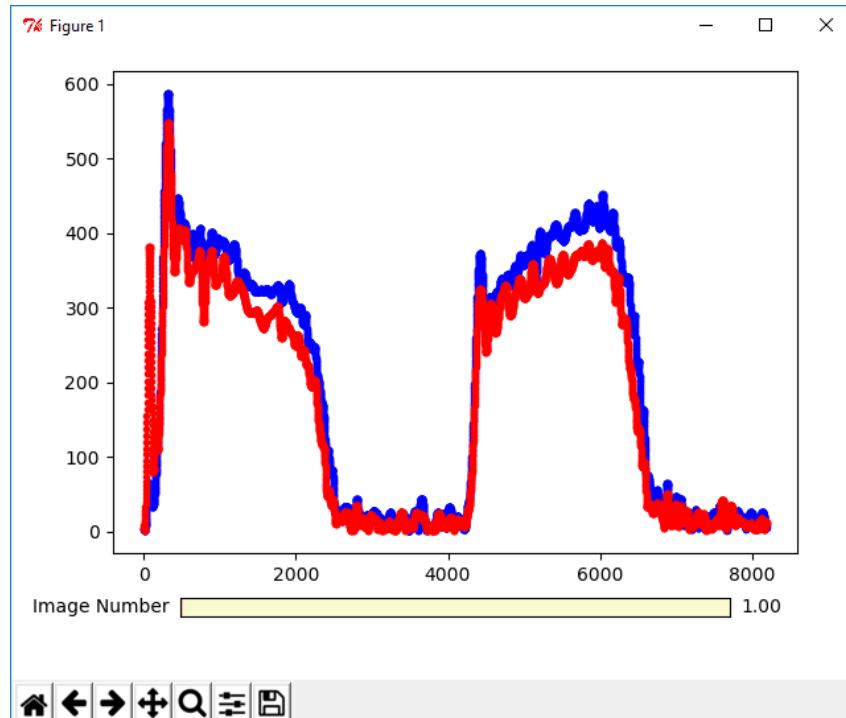
## 4.2 Manipulating Data

This section is based mainly around the code listed under "Hilbert Transform on Saved Data" in appendix A.2.

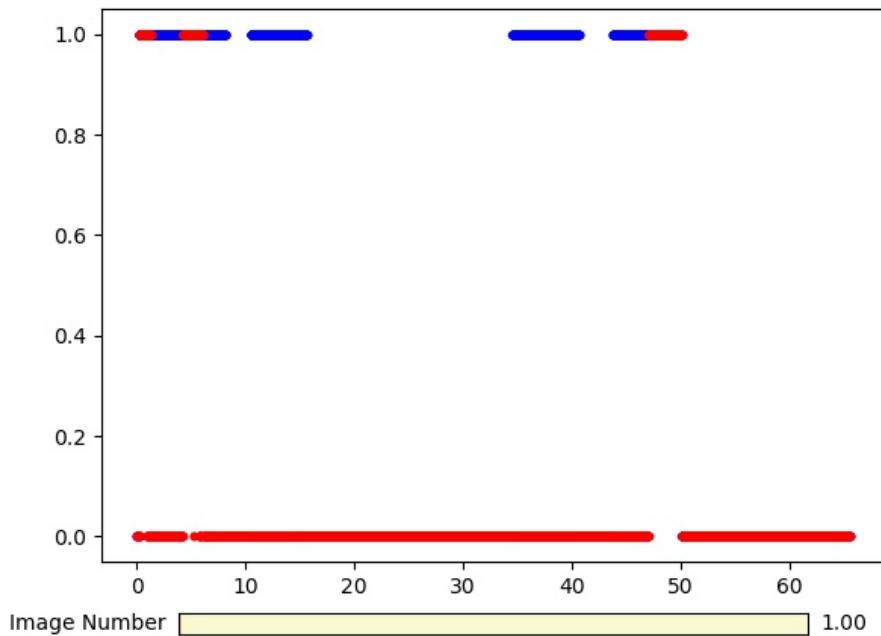
Raw data has been collected from the RP as quickly as possible. This has now left many datasets saved in a CSV format. By selecting the folder location of the datasets, the user will be asked how many csv's they want to analyse. This is just a feature to limit compiling time to look for small details. The code works by first band passing the dataset by applying a Butterworth filter between specified frequencies. Next the filtered signal is passed through a Hilbert Transform function. The Hilbert transform is a linear operator that takes a real input and extends it into the complex plane. It is useful for defining instantaneous properties of functions e.g. Amplitude and frequency.



**Figure 4.1.** The instantaneous amplitude of a function, as generated from the Hilbert transform. This is simply the modulus of the transformed signal. It shows the envelope of the signal.



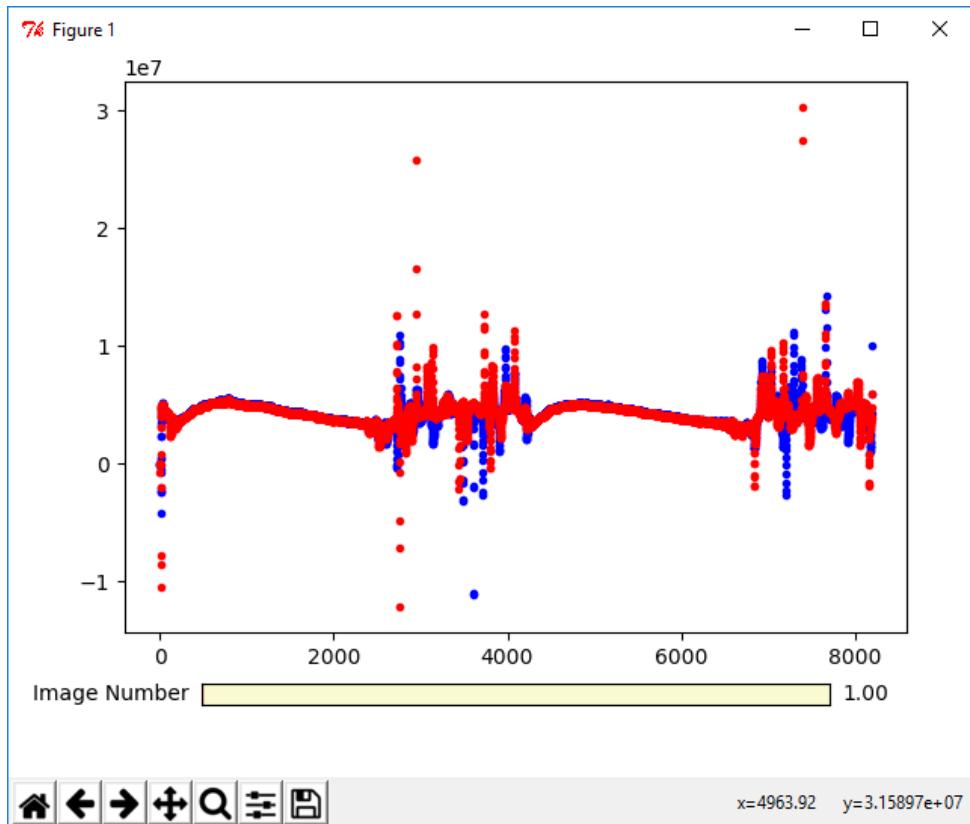
**Figure 4.2.** The instantaneous amplitude of a function, as generated from the Hilbert transform. This is simply the modulus of the transformed signal. It shows the envelope of the signal.



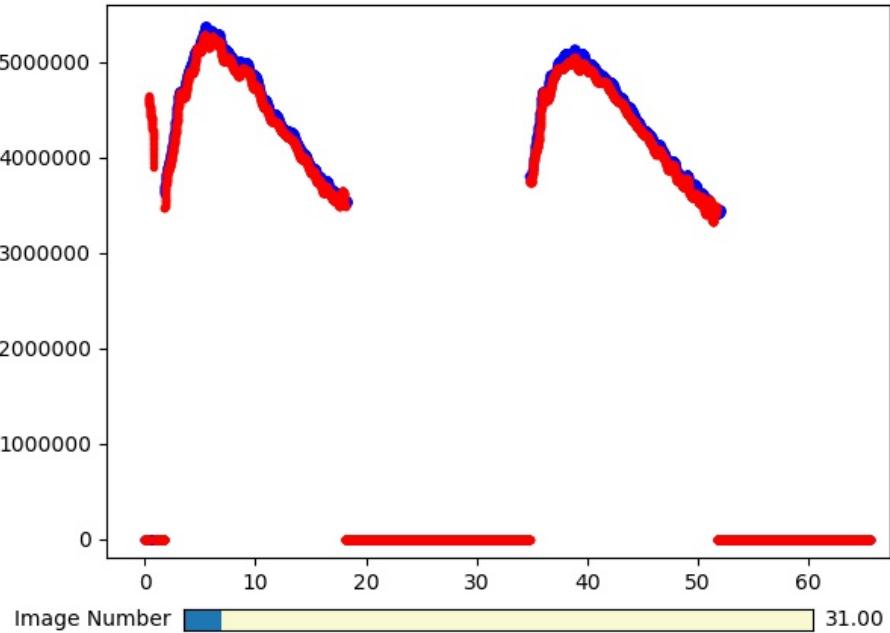
**Figure 4.3.** The threshold instantaneous amplitude around the mean of the instantaneous amplitude. This can now be used as an operator to isolate only the regions of clear information.

As shown in figure 4.1, the data has been filtered around the fringe frequency. This has

eliminated the approximate region of the chirp. As then shown in figure 4.2, the envelope of this filtered signal is then outlined. This is useful, as we can now threshold this to ensure that only specified portions of the signal are analysed. This is done by finding the mean of the instantaneous amplitude and setting any values less than the mean to 0 and greater than the mean to 1. This can then act like an operator to isolate useful portions of the signal. This can be seen in figure 4.3.

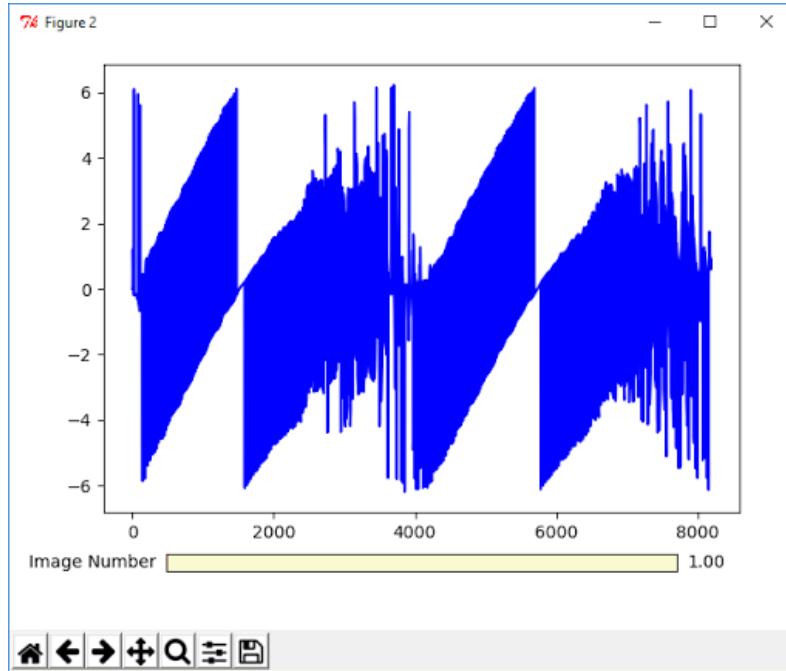


**Figure 4.4.** The instantaneous frequency as obtained by the Hilbert transform. X-Axis is data points against y-axis which is frequency. It is clear to see the bandpass is working, and the region that has been selected is a clear signal.



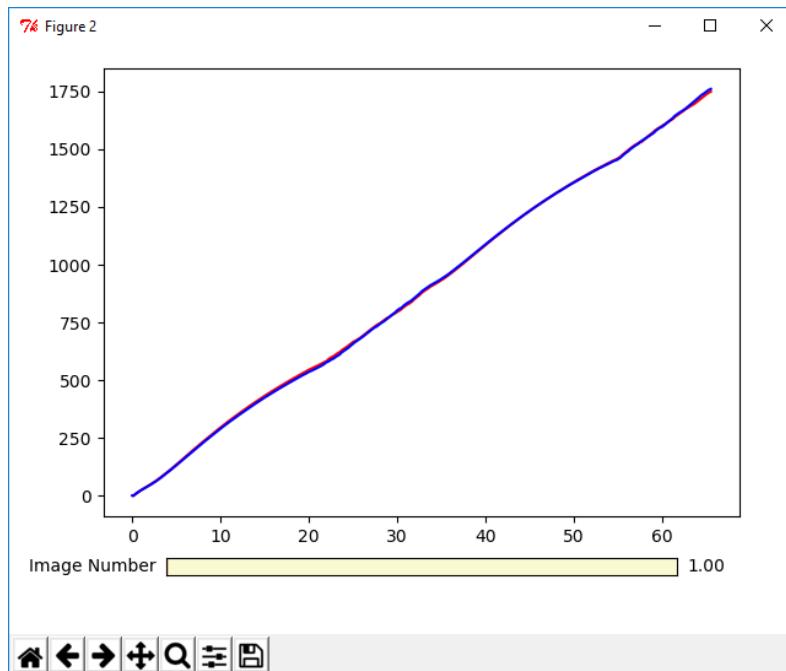
**Figure 4.5.** The instantaneous frequency has now been acted on by the threshold operator. This results in only the regions of clear, slightly varying frequency are shown. All other values appear to be zero.

The Hilbert transform can be used to obtain the instantaneous frequency also by performing a numpy-python function called "diff", which calculated the discrete difference across the signal. The resulting frequency can be seen in figure 4.4. This can be further developed by applying the thresholding operator to completely isolate the clear parts of the signal, as can be seen in figure 4.5. It can also be noted at this point that the fact that these two frequencies are not identical means that there exists a path difference between the two interferometers. For future application this will have to be corrected, but due to the fluctuations in the system a reliable path difference cannot be calculated.



**Figure 4.6**

From the information in the instantaneous frequency the phase can also be obtained from a numpy-python [10] function called "angle". This works by using a arctan function on the real and imaginary parts of the signal. The result will look like a sawtooth wave as it is repeating between  $\pm 2\pi$  degrees as shown in figure 4.6.

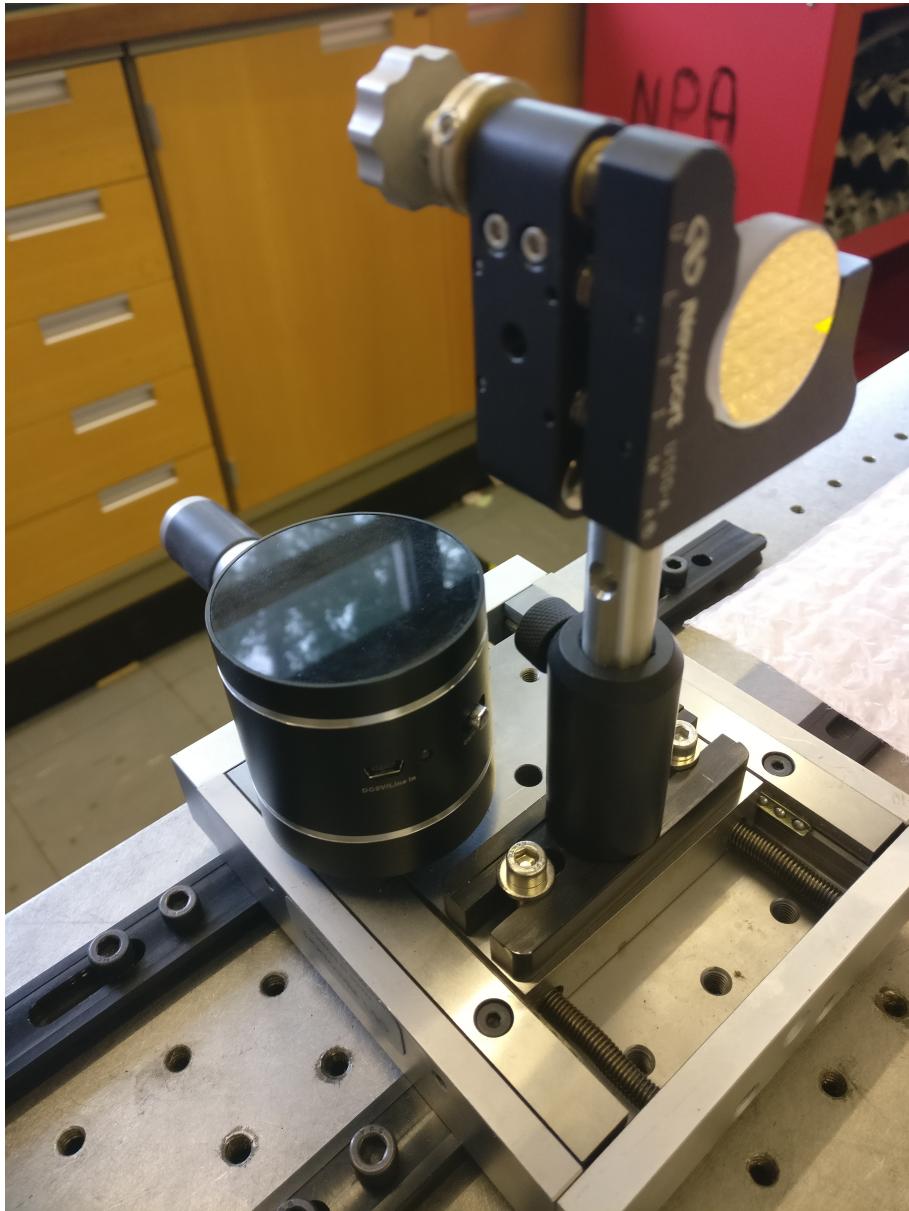


**Figure 4.7.** The unwrapped instantaneous phase of the two channels. A small deviation between the two can be seen. This largely represents the path difference between the two channels. However, this has large fluctuations as a function of time, hence it may also have a large temperature dependence.

By performing another Numpy-python function on the phase, "unwrap", this stops

repeating between  $-2\pi$  and  $+2\pi$ . As a result an accumulative phase is obtained, seen in figure 4.7.

### 4.3 Inducing a phase change by the use of a speaker



**Figure 4.8.** Vibrospeaker (a vibrating speaker) has been placed on the translation stage that is a mount for the mirror in the scene interferometer. By using bluetooth and a signal generator a single frequency can be sent to the speaker to play. This causes it to produce the sound, and hence vibrate at the requested frequency.

By using a vibrospeaker (a vibrating speaker) a frequency can be applied to the mirror in the hopes to add an additional modulation. In order to see this more clearly the Koheron program "Miniscope" was edited to output the unwrapped phase as a dynamic plot, appendix A.3. No visible changes could be seen with the use of the speaker, or the translation stage. It is believed this is because the sampling rate is too high, which is meaning that the frequency of the sound wave is not large enough to be seen on the time scale. To attempt to resolve this data was instead taken with an oscilloscope that had a variable sampling rate. This seemed to still not reveal this underlying vibration that

should be present from the vibrospeaker. Further investigation will be needed to explore the viability of this method as a diagnostic tool.

# Bibliography

- [1] I. H. Hutchinson. *Principles of Plasma Diagnostics*. Cambridge University Press, 2005, p. 460. ISBN: 9780521675741. DOI: 10.1017/CBO9780511613630.
- [2] Kai Jakob Brunner. “FPGA-based High Performance Diagnostics For Fusion”. PhD thesis. University of Durham, Mar. 2017. URL: <http://etheses.dur.ac.uk/12026/>.
- [3] Peter Leban. *Red Pitaya User Manual*. 2014. URL: <http://docs-asia.electrocomponents.com/webdocs/144a/0900766b8144a693.pdf>.
- [4] Koheron. *Laser Board for the Red Pitaya v1*. URL: <https://www.koheron.com/photonics/lbrp-laser-board-red-pitaya-v1>.
- [5] Koheron. *PD100 - Low Noise Photodetector*. URL: <https://www.koheron.com/photonics/pd100-photodetection>.
- [6] Koheron. *A Simple Coherent Laser Sensor*. URL: <https://www.koheron.com/blog/2015/09/10/laser-control>.
- [7] Koheron. *Software Development Kit*. 2016. URL: <https://github.com/Koheron/koheron-sdk>.
- [8] Anaconda Continuum Analytics. *Anaconda Distribution*. 2016. URL: <https://www.continuum.io/>.
- [9] Python Software Foundation. *Python Language Reference, version 2.7*. 2013. DOI: <https://www.python.org/>. URL: <http://www.python.org>.
- [10] NumPy Community. *NumPy 1.7.0*. 2011. URL: <http://www.numpy.org/>.
- [11] Pearu Peterson Eric Jones Travis Oliphant et al. {SciPy}: *Open source scientific tools for {Python}*. 0. URL: <http://www.scipy.org/>.

# Code

## A.1 Saving Data

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Prior to running Koheron must be installed to the python
    directory
5  #If using anaconda: anaconda prompt -> "pip install koheron"
6  from koheron import connect
7  from oscillo import Oscillo
8
9  import sys
10 import os
11 import numpy as np
12 import math
13 import csv
14
15 from Tkinter import Tk
16 import tkFileDialog
17
18 import matplotlib
19 matplotlib.use('TKAgg')
20 from matplotlib import pyplot as plt
21
22 #IP Address of the Red Pitaya (RP) can vary. It should show on
    the RP by yellow LED's
23 #each corresponding to a base-2 number. e.g. Two to the power of
    the number of the LED.
24 #If this is not functional, go to the commandprompt and type "
    nslookup Koheron"
25 host = os.getenv('HOST', '10.209.1.125')
26 #opens a connection to the clientside of the application, in
    this case, oscillo.
27 client = connect(host, name='oscillo')
28
29 #Select a folder where you wish to save the data.
30 Tk().withdraw() # keep the root window from appearing
```

```

31 # show an "Open" dialog box and return the path to the selected
32     file
33     filepath = tkFileDialog.askdirectory(title="Select A Folder",
34                                         mustexist=1)
35     print(filepath)
36
37
38
39     driver = Oscillo(client)
40     driver.reset()
41     t = 1e6*driver.t
42
43 #initialise channel0 and 1 arrays for data collection.
44 #driver.wfm_size = 8192 data points = length of FIFO.
45     channel0 = np.zeros((loop, driver.wfm_size))
46     channel1 = np.zeros((loop, driver.wfm_size))
47
48 # Modulation on DAC
49 #Amplitude of the modulation
50     amp_mod = 0.995
51 #Frequency of the modulation. driver.sampling_rate = 125e6
52     freq_mod = driver.sampling_rate / driver.wfm_size * 10
53 #Input modulation for channel 0 and 1 accross all data points
54     resepectively.
55     driver.dac[0, :] = amp_mod*np.cos(2 * np.pi * freq_mod * driver.
56                                         t)
57     driver.dac[1, :] = amp_mod*np.sin(2 * np.pi * freq_mod * driver.
58                                         t)
59     driver.set_dac()
60
61     time.sleep(0.001)
62
63
64
65
66 #Set averaging for the dataset. If false , no averaging is
67     performed.
68     driver.set_num_average_min(2000)
69     driver.set_average(False)
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2
```

```
72 #write data 1 row at a time into csv
73 for i in range(0, loop):
74     write_file = filepath+ '\\'+ str(i) + '.csv'
75     with open(write_file, "wb") as csv_file:
76         csv_writer = spamwriter = csv.writer(csv_file,
77                                             delimiter=',', quoting=csv.QUOTE_MINIMAL)
78         csv_writer.writerow(['Time(us)', 'Channel 0',
79                             'Channel 1'])
80         for j in range(0, channel0[i].size):
81             csv_writer.writerow([t[j], channel0[i][j],
82                                 channel1[i][j]])
83     csv_file.close()
```

## A.2 Hilbert Transform on Saved Data

```
1 import os
2 import numpy as np
3 import sys
4 import math
5
6 import matplotlib
7 from matplotlib import mlab
8 matplotlib.use('TKAgg')
9 from matplotlib import pyplot as plt
10 from matplotlib.widgets import Slider
11
12
13 from scipy import signal
14 from scipy.signal import hilbert, chirp
15 from scipy.signal import butter, lfilter, filtfilt
16
17 from Tkinter import Tk
18 import tkFileDialog
19 import tkSimpleDialog
20
21 #pop up window which allows easier locating of folders
22 Tk().withdraw() # keep the root window from appearing
23 ## show an "Open" dialog box and return the path to the selected
24 file
25 filepath = tkFileDialog.askdirectory(title="Select A Folder",
26 mustexist=1)
27 print(filepath)
28
29 #Works out how many files are in a folder.
30 path, dirs, files = os.walk(filepath).next()
31 #take 2 as these represent subdirectories
32 n_files = len(files) - 2
33 print(n_files)
34
35 #allows a selection of number of files wanting to be analysed
36 n_files_sel = int(tkSimpleDialog.askstring("Data Popup", "There
37 are "+str(n_files)+" in this folder, how many would you like
38 to use?"))
39
40 channel0 = np.zeros((n_files_sel, 8192))
41 channel1 = np.zeros((n_files_sel, 8192))
42
43 #only collects columns 2 and 3 so as to save time. Column 3
44 #should be identical in all cases because it is the timebase.
45 for i in range(1, n_files_sel+1):
46     csv_data = np.hsplit((np.genfromtxt(filepath+"//"+str(i)
```

```

-1)+".csv", delimiter=",", dtype=float, skip_header=
        True)),3)
42 channel0[i-1] = np.ravel(csv_data[1])
43 channel1[i-1] = np.ravel(csv_data[2])
44 t = np.ravel(csv_data[0])
45
46
47 class DiscreteSlider(Slider):
48     """A matplotlib slider widget with discrete steps."""
49     def __init__(self, *args, **kwargs):
50         """
51             Identical to Slider.__init__, except for the new keyword
52             'allowed_vals'.
53             This keyword specifies the allowed positions of the
54             slider
55         """
56         self.allowed_vals = kwargs.pop('allowed_vals',None)
57         self.previous_val = kwargs['valinit']
58         Slider.__init__(self, *args, **kwargs)
59         if self.allowed_vals==None:
60             self.allowed_vals = [self.valmin, self.valmax]
61
62     def set_val(self, val):
63         discrete_val = self.allowed_vals[abs(val-self.
64             allowed_vals).argmin()]
65         xy = self.poly.xy
66         xy[2] = discrete_val, 1
67         xy[3] = discrete_val, 0
68         self.poly.xy = xy
69         self.valtext.set_text(self.valfmt % discrete_val)
70         if self.drawon:
71             self.ax.figure.canvas.draw()
72         self.val = val
73         if self.previous_val!=discrete_val:
74             self.previous_val = discrete_val
75             if not self.eventson:
76                 return
77             for cid, func in self.observers.iteritems():
78                 func(discrete_val)
79
80     #Slider plot used to compare two data sets with defined x axis
81     def slider_plot(x_data, c0_data, c1_data, c0_args, c1_args):
82         axcolor = 'lightgoldenrodyellow'
83         image_spacing = np.linspace(1, n_files_sel, n_files_sel)
84         t_sp = x_data
85         fig, ax = plt.subplots()
86         plt.subplots_adjust(left=0.12, bottom=0.2, right=0.93,
87             top=0.96, wspace=0.2, hspace=0.2)

```



```

124     l, = plt.plot(t[0:c0_data[0].size], c0_data[0], c0_args)
125     ax_img_number = plt.axes([0.2, 0.1, 0.65, 0.03],
126                               facecolor=axcolor)
127     s_img = DiscreteSlider(ax_img_number, 'Image Number', 1,
128                            n_files_sel, valinit=1, allowed_vals=image_spacing)
129     def updateplot(val):
130         fig_number = int(s_img.val)
131         l.set_ydata(c0_data[fig_number-1])
132         fig.canvas.draw_idle()
133     s_img.on_changed(updateplot)
134
135 #####~~~~~Try to calculate the phase between the two channels
136 #####~~~~~#####
137 #function to perform a bandpass between selected frequencies
138 def bandpass_data(t_in, data_in, fmin, fmax):
139     tt=t_in
140     del_t=tt[1]-tt[0]
141     fs=1/(del_t)
142     highf=fmax / (fs / 2)
143     lowf=fmin/(fs/2)
144     B, A = butter(5, [lowf, highf], btype='bandpass')
145     filtered_signal = lfilter(B, A, data_in, axis=0)
146     return filtered_signal
147
148 #performs a hilbert transform returning amplitude, frequency
149 # and phase
150 def hilbert_signal_phase(signal, fs):
151     #if signal.size < 2*22:
152     #    analytic_signal = hilbert(signal)
153     #else:
154     #    analytic_signal = hilbert(signal, N=2**22)
155     analytic_signal = hilbert(signal)
156     analytic_signal=analytic_signal[0:signal.size]
157     amplitude_envelope = np.abs(analytic_signal)
158     instantaneous_phase = np.unwrap(np.angle(analytic_signal))#,
159     #2*np.pi)
160     instantaneous_frequency = np.diff.instantaneous_phase) /
161     (2.0*np.pi) * fs
162     #return amplitude_envelope, instantaneous_frequency,
163     #instantaneous_phase
164     return amplitude_envelope, instantaneous_frequency,
165     instantaneous_phase
166
167 N = 8191
168 t_min = 0

```

```

164 t_max = N / 125e6
165 t_range = np.linspace(t_min, t_max, N)
166
167
168 sampling = 125e6
169 N_data = 1024
170 n_angle = np.zeros((n_files_sel, 513), dtype=float)
171 csv_filter_s1 = np.zeros((n_files_sel, 8192))
172 csv_filter_s2 = np.zeros((n_files_sel, 8192))
173 k=0
174
175 #perform a bandpass on the data between the desired intervals
176 for i in range(0, n_files_sel):
177     csv_filter_s1[i]=bandpass_data(t_range, channel0[i], 3.5
178                                     e6, 5.5e6)
179     csv_filter_s2[i]=bandpass_data(t_range, channel1[i], 3.5
180                                     e6, 5.5e6)
181
182 #perform hilbert transform on the now filtered data
183 AE1, IF1, Ph1 = hilbert_signal_phase(csv_filter_s1, 125e6)
184 AE2, IF2, Ph2 = hilbert_signal_phase(csv_filter_s2, 125e6)
185
186 #Threshold the amplitude against its mean. This value can now be
187 #used to remove areas of non-interest by multiplying any
188 #array by TAE1 or 2
189 phase = np.zeros((n_files_sel, 8192))
190 TAE1 = np.zeros(AE1)
191 TAE2 = np.zeros(AE2)
192 for i in range(0, n_files_sel):
193     threshold_AE1 = np.mean(AE1[i])
194     threshold_AE2 = np.mean(AE2[i])
195     for j in range(0, 8192):
196         if np.mean(AE1[i][j])<threshold_AE1:
197             TAE1[i][j] = 0
198         else:
199             TAE1[i][j] = 1
200         if np.mean(AE2[i][j])<threshold_AE2:
201             TAE2[i][j] = 0
202         else:
203             TAE2[i][j] = 1
204
205 phase = (Ph1-Ph2)*TAE1
206
207 sys.exit

```

### A.3 Hilbert Transform in Real-time

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 # Function generator with live acquisition plot for RedPitaya
5
6 import os
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from scipy import signal
10 from laser import Laser
11
12 from scipy import signal
13 from scipy.signal import hilbert
14 from scipy.signal import butter, lfilter
15
16 from koheron import connect
17 from oscillo import Oscillo
18
19 #Bandpasses the data between fmin and fmax
20 def bandpass_data(t_in, data_in, fmin, fmax):
21     tt=t_in
22     del_t=tt[1]-tt[0]
23     fs=1/(del_t)
24     highf=fmax / (fs / 2)
25     lowf=fmin/(fs/2)
26     B, A = butter(5, [lowf, highf], btype='bandpass')
27     filtered_signal = lfilter(B, A, data_in, axis=0)
28     return filtered_signal
29
30 #perfromes a hilbert transform returning amplitude, frequency
31 # and phase
32 def hilbert_signal_phase(signal, fs):
33     #if signal.size < 2*22:
34     #    analytic_signal = hilbert(signal)
35     #else:
36     #    analytic_signal = hilbert(signal, N=2**22)
37     analytic_signal = hilbert(signal)
38     analytic_signal=analytic_signal[0:signal.size]
39     amplitude_envelope = np.abs(analytic_signal)
40     instantaneous_phase = np.unwrap(np.angle(analytic_signal))#,
41     #2*np.pi)
42     #instantaneous_phase = np.angle(analytic_signal)#, 2*np.pi)
43     instantaneous_frequency = np.diff(instantaneous_phase) /
44     (2.0*np.pi) * fs
45     #return amplitude_envelope, instantaneous_frequency,
46     #instantaneous_phase
```

```

43     return amplitude_envelope, instantaneous_frequency ,
44           instantaneous_phase
45
45 host = os.getenv('HOST', '10.209.1.125')
46 client = connect(host, name='oscillo')
47 driver = Oscillo(client)
48
49 #Starts the laser output at selected current
50 Laser(client).start()
51 Laser(client).set_current(30)
52
53 n = driver.wfm_size # Number of samples
54 fs = 125E6           # Sampling rate (Hz)
55 n_bits = 14          # ADC bits
56
57 #


---


58 # Waveform generation
59 #


---


60
61 # Modulation on DAC
62
63 #averaging settings
64 driver.set_average(False)
65 driver.set_num_average_min(0)
66
67 #((channel 1, channel 2) (0=sine, 1=triangle, 2=square)
68 driver.set_waveform_type(1, 1)
69
70 #modulation on each channel
71 driver.set_dac_amplitude(1, 0.8)
72 driver.set_dac_frequency(1, 125e6/8192)
73 driver.set_dac_offset(1, 0.0)
74
75 #Modulation only seems to apply after "pinging" the pitaya.
76 driver.get_modulation_status()
77
78 #


---


79 # Data acquisition
80 #


---


81

```

```

82 adc_channel = 0
83
84 #driver.set_average(True)
85
86 fig = plt.figure()
87 ax = fig.add_subplot(111)
88 t = np.arange(n) / fs # Time axis in us
89 y = np.zeros(n)
90 li1, = ax.plot(t[:8191], y[:8191])
91 li2, = ax.plot(t[:8191], y[:8191])
92 ax.set_ylim(-20,20)
93 ax.set_xlim(2.5e-6,15e-6)
94 ax.set_xlabel('Time (us)')
95 ax.set_ylabel('Signal (V)')
96 fig.canvas.draw()
97 plt.show(block=False)
98
99 csv_filter_s1 = np.zeros((8192))
100 csv_filter_s2 = np.zeros((8192))
101
102 while True:
103     try:
104         #collects data from the Red Pitaya
105         driver.get_adc()
106
107         #Filters the data between the given frequencies
108         csv_filter_s1=bandpass_data(t, driver.adc[0,:]/
109                                     2**n_bits,3.5e6,5.5e6)
110         csv_filter_s2=bandpass_data(t, driver.adc[1,:]/
111                                     2**n_bits,3.5e6,5.5e6)
112
113         #Performs a hilbert tranform on the filtered
114         # data
115         AE1, IF1, PH1 = hilbert_signal_phase(
116             csv_filter_s1, 125e6)
117         AE2, IF2, PH2 = hilbert_signal_phase(
118             csv_filter_s2, 125e6)
119
120         #Thresholds the data using the envelope given by
121         # the hilbert
122         TAE1 = AE1
123         TAE2 = AE2
124
125         threshold_AE1 = np.mean(AE1[i])
126         threshold_AE2 = np.mean(AE2[i])
127         for j in range(0, datapoints):
128             if np.mean(AE1[j])<threshold_AE1:
129                 TAE1[j] = 0

```

```

124
125         else :
126             TAE1[ j ] = 1
127             if np.mean(AE2[ j ]) < threshold_AE2 :
128                 TAE2[ j ] = 0
129             else :
130                 TAE2[ j ] = 1
131             #note frequency has only 8191 points , where as
132             #all others have 8192.
133             #Uncomment the appropriate plot for the purpose
134             # of the data.
135
136
137             #shows the phase of each channel
138             #li1.set_ydata(PH1[:8191])
139             #li2.set_ydata(PH2[:8191])
140
141             #shows the envelope of each channel
142             #li1.set_ydata(AE1[:8191])
143             #li2.set_ydata(AE2[:8191])
144
145             #shows the frequency of each channel
146             #li1.set_ydata(IF1[:8191])
147             #li2.set_ydata(IF2[:8191])
148
149             #shows the thresholded phase of each channel
150             #li1.set_ydata(PH1[:8191]*TAE1[:8191])
151             #li2.set_ydata(PH2[:8191]*TAE2[:8191])
152
153             #shows the thresholded envelope of each channel
154             #li1.set_ydata(AE1[:8191]*TAE1[:8191])
155             #li2.set_ydata(AE2[:8191]*TAE2[:8191])
156
157             #shows the thresholded frequency of each channel
158             #line.remove(li2)
159             #li1.set_ydata(PH1[:8191]*TAE1[:8191] - PH
160             #[:8191]*TAE2[:8191])
161
162             #shows the thresholded difference in phase of
163             # each channel
164             #line.remove(li2)
165             #li1.set_ydata(IF1[:8191]*TAE1[:8191] - IF2

```

```
166 #li1.set_ydata(csv_filter_s1[:8191])
167 #li2.set_ydata(csv_filter_s2[:8191])
168
169 #shows the raw data of each channel
170 #li1.set_ydata(driver.adc[0,:][:8191])
171 #li2.set_ydata(driver.adc[1,:][:8191])
172
173 ax.relim()
174 ax.autoscale_view(True, True, True)
175
176 plt.pause(0.001)
177 fig.canvas.draw()
178
179 except KeyboardInterrupt:
180     break
```