recon_fft

January 21, 2025

```
In [1]: from pynq import Overlay
        from math import log

In [2]: ol = Overlay('recon_fft.bit')

In [3]: ol?

In [4]: data_dma = ol.fft_block.data_dma
        config_dma= ol.fft_block.config_dma

In [5]: data_send =data_dma.sendchannel
        data_recv= data_dma.recvchannel
        conf_send = config_dma.sendchannel
```

1 Creating Configuration

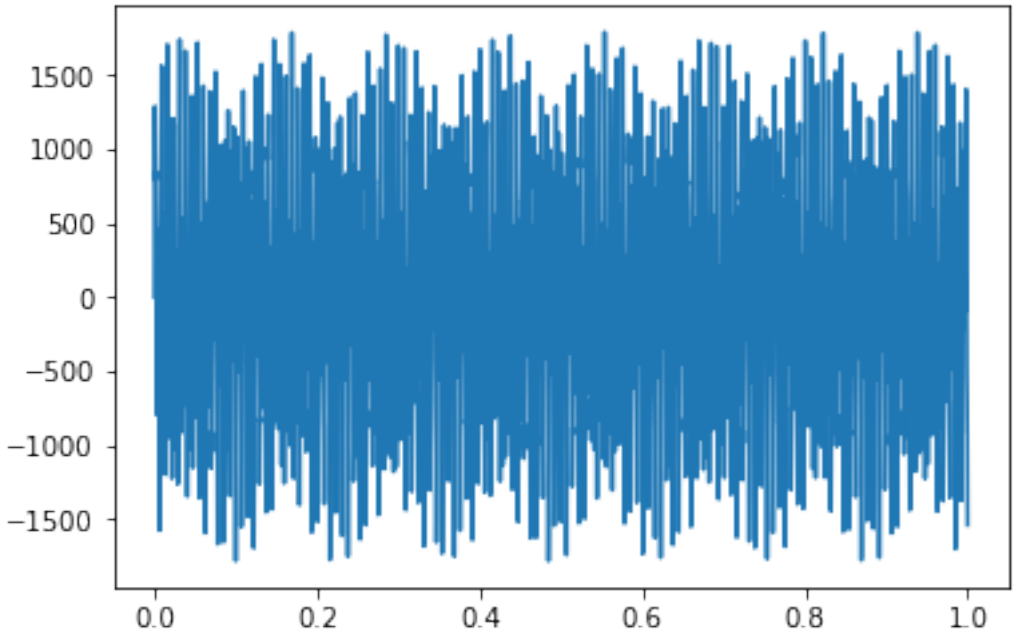
```
In [6]: #Pad all values till they are multiple of 8 bits
        #Last 5 bits select FFT size
        #We can set the values for the second last byte as all zeroes
        #The most significant byte to set IP as FFT or inverse FFT, we set its value to 1 as h

        def convert_to_data(fft_direction,size):
            fft_direction.zfill(8) # '011'-> 0000 0011

            byte2 = '0'*8

            x= int(log(size,2))
            fft_size = bin(x) [2:] # convert it into binary value

            fft_size.zfill(8)
            tdata= fft_direction+byte2+fft_size
            return int(tdata,2)



In [7]: import numpy as np
        import matplotlib.pyplot as plt
        import random
```

In [8]:
```
SAMPLES = 8192 # In case no config is provided FFT defaults to 8192 transform size and
```

In [9]:
```
time_interval = 1
def create_data(SAMPLES,time_interval):
    A1= random.uniform(100,1000)
    A2= random.uniform(100,1000)
    A3 = random.uniform(100,1000)
    f1= random.uniform(100,150)
    f2= random.uniform(200,300)
    f3= random.uniform(500,600)
    w1= 2*np.pi*f1
    w2= 2*np.pi*f2
    w3= 2*np.pi*f3
    t= np.linspace(0,time_interval,SAMPLES)
    data= A1*np.sin(w1*t,dtype=np.csingle) + A2*np.sin(w2*t,dtype=np.csingle)+ A3*np.s
    return data,t
```

In [10]:
```
data,t= create_data(SAMPLES,time_interval)
plt.plot(t,np.real(data))
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0xaeda5cd0>]
```



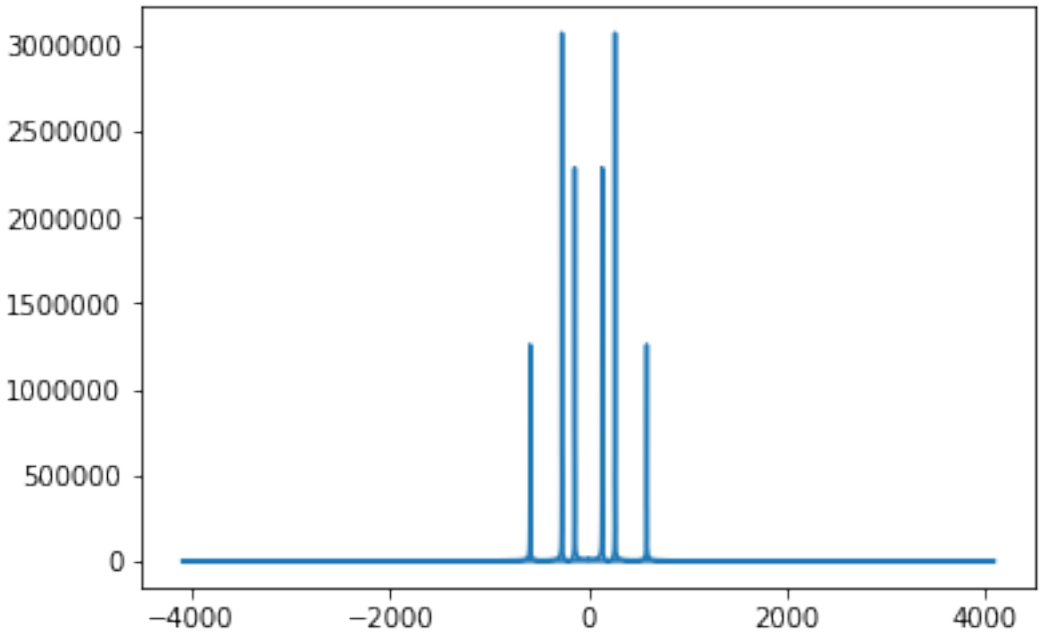
In [11]:
```
%%time
import time
sw_start= time.time()
output = np.fft.fft(data)
```

```
        freq= np.fft.fftfreq(SAMPLES*time_interval,1/SAMPLES)
        sw_end= time.time()
        sw_exec_time = sw_end- sw_start
```

```
CPU times: user 10.7 ms, sys: 55 µs, total: 10.8 ms
Wall time: 10.9 ms
```

```
In [12]: plt.plot(freq,np.abs(output))
```

```
Out[12]: [<matplotlib.lines.Line2D at 0xaec25430>]
```

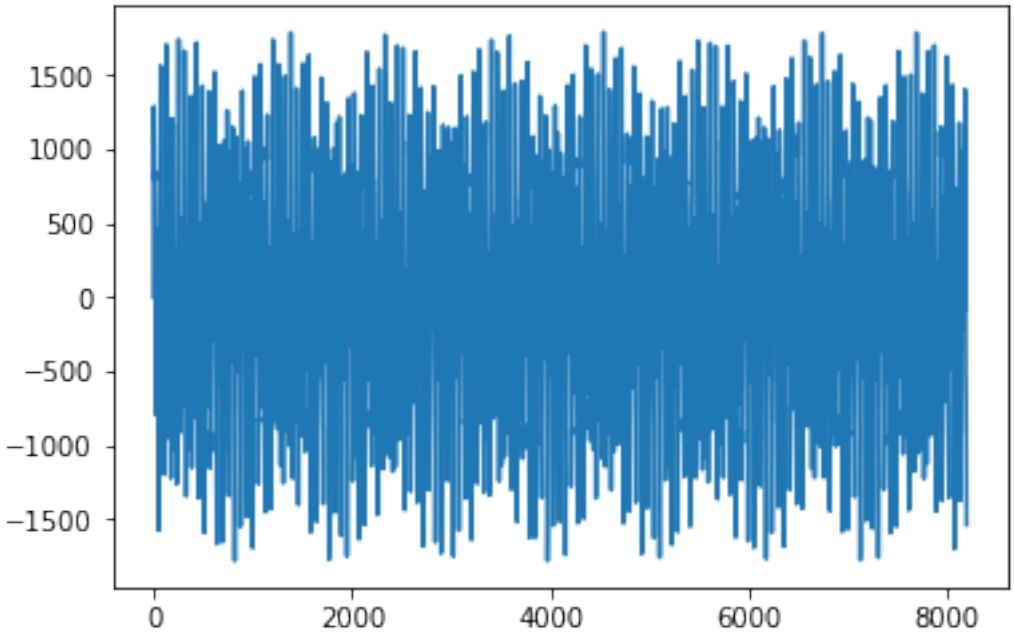



```
In [13]: from pynq import allocate
```

```
In [14]: input_buffer= allocate(SAMPLES,np.csingle)
         output_buffer= allocate(SAMPLES,np.csingle)
         np.copyto(input_buffer,data)
```

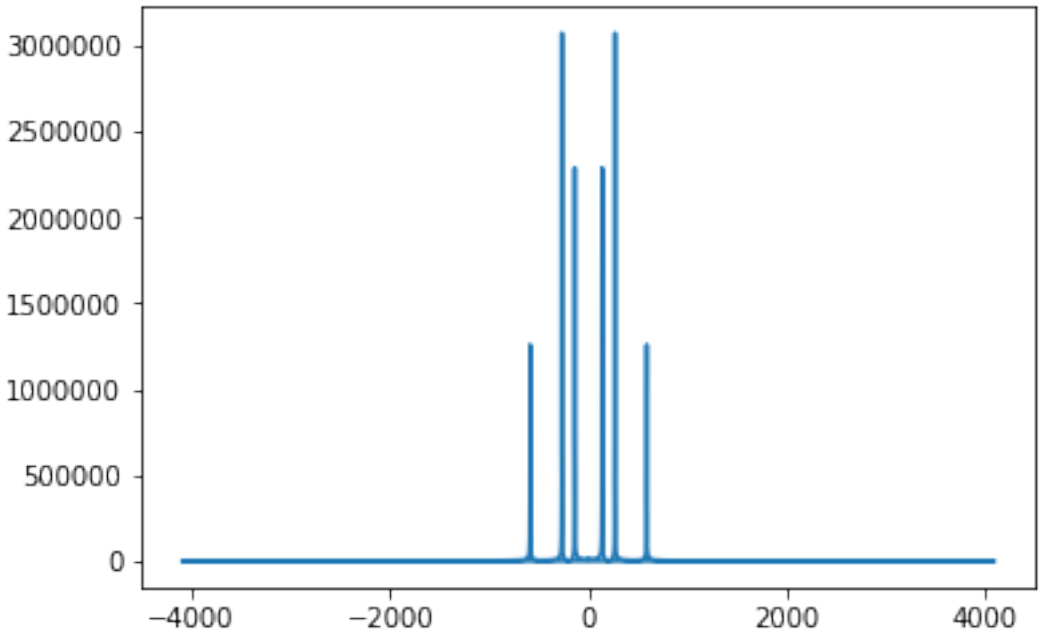
```
In [15]: plt.plot(np.real(input_buffer))
```

```
Out[15]: [<matplotlib.lines.Line2D at 0xaed15f90>]
```




```
In [16]: %%time
         import time
         hw_start = time.time()
         data_send.transfer(input_buffer)
         data_recv.transfer(output_buffer)
         data_send.wait()
         data_recv.wait()
         hw_end = time.time()
         hw_exec_time = hw_end-hw_start

CPU times: user 705 µs, sys: 42 µs, total: 747 µs
Wall time: 798 µs


In [17]: output_buffer

Out[17]: PynqBuffer([ 10296.56250  +0.j      ,  10296.09375 +56.09375j,
                      10297.56250+113.78125j, ...,  10299.09375-170.6875j ,
                      10297.59375-113.96875j,  10296.12500 -56.28125j], dtype=complex64)

In [18]: plt.plot(freq,np.abs(output_buffer))
         print("Hardware exec time :",hw_exec_time)
         print("Software exec time :",sw_exec_time)
         print("Hardware acceleration factor",sw_exec_time/hw_exec_time)

Hardware exec time : 0.0006735324859619141
Software exec time : 0.010706901550292969
```

```
Hardware acceleration factor 15.896637168141593
```

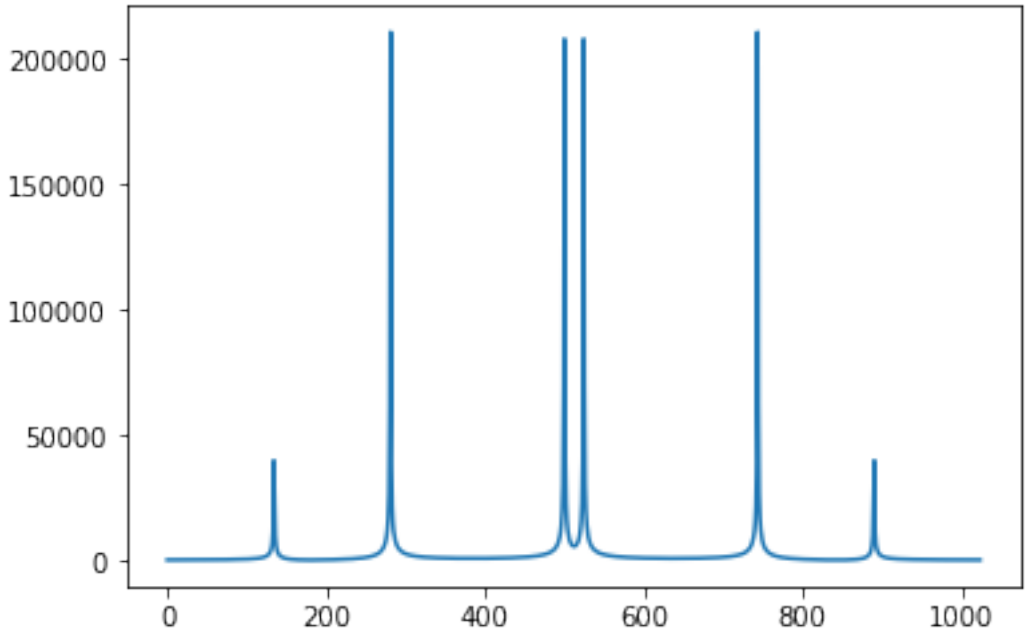



```
In [19]: SAMPLES =1024 # changing fft size
         data,t= create_data(SAMPLES,time_interval)

In [20]: conf_buffer= allocate(1,np.uint32) #buffer of size 1 which send a uint 32 value
         conf_buffer[0]= convert_to_data('1',SAMPLES)

In [21]: conf_send.transfer(conf_buffer)
         conf_send.wait()

In [22]: input_buffer1= allocate(SAMPLES,np.csingle)
         output_buffer1= allocate(SAMPLES,np.csingle)
         np.copyto(input_buffer1,data)

In [23]: %%time
         data_send.transfer(input_buffer1)
         data_recv.transfer(output_buffer1)
         data_send.wait()
         data_recv.wait()

CPU times: user 1.02 ms, sys: 60 µs, total: 1.08 ms
Wall time: 1.24 ms


In [24]: output_buffer1
```

```
Out[24]: PynqBuffer([ 335.17773438+0.j        ,  335.14453125-0.328125j  ,
                      336.93554688+1.92773438j, ...,  336.62109375+0.9765625j ,
                      336.93359375-1.9375j    ,  335.14453125+0.31835938j], dtype=complex64)
```

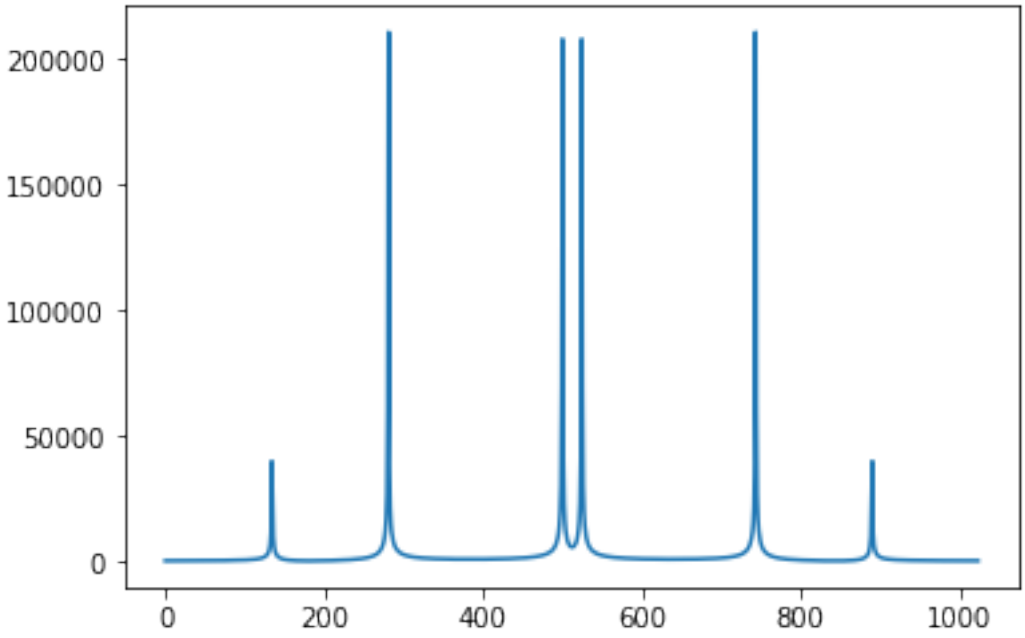
```
In [25]: plt.plot(np.abs(output_buffer1))
```

```
Out[25]: [<matplotlib.lines.Line2D at 0xaecd96d0>]
```




```
In [26]: plt.plot(np.abs(np.fft.fft(data)))
```

```
Out[26]: [<matplotlib.lines.Line2D at 0xaec97850>]
```


200000
150000
100000
50000
0
0
200
400
600
800
1000