In [28]:
```
import numpy as np
import random
from pynq import Overlay
import axitimer
max_fft_N = 12
```

In [30]:
```
ol = Overlay('FFTPynq.bit', ignore_version=True)
```

In [31]:
```
print(ol.ip_dict.keys())
```

```
dict_keys(['axi_dma_0', 'axi_intc_0', 'axi_timer_0', 'axi_fifo_mm_s_0', 'proce
ssing_system7_0'])
```

In [32]:
```
tmr = ol.axi_timer_0
tmr.start_tmr()
sw_fft_times = []
for i in range(6,max_fft_N+1):
    swt = 0

    size = 2 ** i
    din = np.empty(shape=(size,), dtype='complex64')
    din.imag[0:size] = np.random.rand(size,)
    din.real[0:size] = np.random.rand(size,)
    din[20] = 1.0;
    for n in range(0,100):
        start = tmr.read_count()
        dout_sw = np.fft.fftn(din)
        end = tmr.read_count()
        swt += tmr.time_it(start,end)
    sw_fft_times.append([size, 1000*swt/(n+1)])
print('[size, Time in mS]')
print(sw_fft_times)
```

```
[size, Time in mS]
[[64, 0.1405303201873738], [128, 0.14646230686194972], [256, 0.1722593135630
124], [512, 0.21928489362571318], [1024, 0.30967491374623324], [2048, 0.5395
103940526804], [4096, 1.077804328103739]]
```

In [33]:
```
def bits_to_bytes(bit_str, base=2, byteorder='big') :
    nbytes = int(len(bit_str) * .125 + .9) if base == 2 else int(len(bit_str) *
    return int(bit_str, base=base).to_bytes(nbytes, byteorder=byteorder)

def bytes_to_uint32s(byte_s) :
    assert type(byte_s) == bytes, RuntimeError
    return [int.from_bytes(byte_s[i:i+4], byteorder='big') for i in range(0,len

def create_config_tdata(N, fwd_inv=True):
    assert N > 2 and N < 13, RuntimeError
    N = int(N)
    NFFT = N.to_bytes(1, byteorder='big')            # Must be padded to byte
    CP_LEN = ''                                      # padded but unused for th
    FWD_INV = '0b1' if fwd_inv == True else '0b0'    # No padding
    SCALE_SCH = ''                                   # padded but unused for th
    return bytes_to_uint32s(bits_to_bytes(FWD_INV) + NFFT) # Final result needs
```

In [35]:
```
# The constraints say it should work up to about 150MHz
from pynq import Clocks
```

In [36]:
```
from axififo import FifoStreamDriver
ol = Overlay('FFTPynq.bit', download=False, ignore_version=True)
cfg_data = create_config_tdata(max_fft_N, True)
fft_cfg = ol.axi_fifo_mm_s_0
# Assume 'cfg_data' is a list or array of data to send
for word in cfg_data:
    ol.axi_fifo_mm_s_0.write(0x00, word)  # Replace <offset> with the appropri

# If the FIFO requires triggering or start signals, you might need to write to
```

In [37]:
```
import axidma
ol = Overlay('FFTPynq.bit', download=False, ignore_version=True)
fft = ol.axi_dma_0
fft.resize_bufs(shape=[size,], dtype='complex64')
fft.txbuf[0:size] = din
fft.send_dma()
fft.rcv_dma()
fft.rxbuf
```

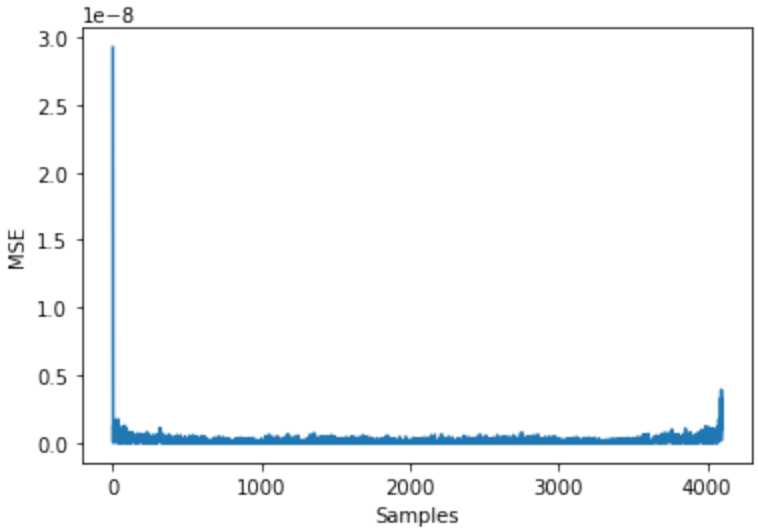
Out[37]:
```
PynqBuffer([2034.578    +2051.871j    ,   23.957458    -4.518738j ,
               6.3262024  -32.044525j , ...,   -2.402298   +52.793182j ,
            -15.237122    -6.6253815j,   -3.3567352  -40.571396j ],
          dtype=complex64)
```

In [38]:
```
np.fft.fftn(din)
```

Out[38]:
```
array([2034.5780469 +2051.87118637j,   23.95752363   -4.51868644j,
         6.32625788  -32.04449238j, ...,   -2.40226335  +52.79324482j,
       -15.23707777   -6.62532685j,   -3.35668979  -40.57132555j])
```

In [27]:
```
%matplotlib inline
import matplotlib.pyplot as plt
hw_fft_data = np.abs(np.array(fft.rxbuf))
sw_fft_data = np.abs(np.array(np.fft.fftn(din)))
err_squared = (hw_fft_data-sw_fft_data)**2
f, ax = plt.subplots()
ax.plot(err_squared)
ax.set_xlabel('Samples')
ax.set_ylabel('MSE');
print('mean squared error: ' + str(err_squared.mean()))
```

```
mean squared error: 1.563096694650876e-10
```



In [ ]:
```
sw_fft_times = []
hw_fft_times = []
tmr.start_tmr()
for i in range(6,max_fft_N+1):
    hwt = swt = 0
    size = 2 ** i
    din = np.empty(shape=(size,), dtype=np.complex64)
    din.imag[0:size] = np.random.rand(size,)
    din.real[0:size] = np.random.rand(size,)
    fft_cfg.send_tx_pkt(create_config_tdata(i, 1))
    for n in range(0,100):
        start = tmr.read_count()
        dout_sw = np.fft.fftn(din)
        end = tmr.read_count()
        swt += tmr.time_it(start,end)
        start = tmr.read_count()
        fft.send_cpy(din)
        dout = fft.rcv_cpy(size, np.complex64)
        end = tmr.read_count()
        hwt += tmr.time_it(start,end)
    sw_fft_times.append([size, 1000*swt/(n+1)])
    hw_fft_times.append([size, 1000*hwt/(n+1)])
print('Times are in mS')
print(sw_fft_times)
print(hw_fft_times)
print('Acceleration factor:')
a = []
for i in range(0,len(hw_fft_times)):
    a.append([sw_fft_times[i][0], sw_fft_times[i][1]/hw_fft_times[i][1]])
print(a)
```