In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
#here i genereate 1,000,000 data points using the generation model
n_sample = 1000000
gn = [np.random.normal(),np.random.normal(),np.random.normal()]
data = []
for idx in range(3,n_sample+3):
    g = np.random.normal()
    gn.append(g)
    data.append(.1*g+.5*gn[idx-1]-.5*gn[idx-2]+.1*gn[idx-3]) # x_[n] = u_[n] + 0.8*x_[
```

In [5]:

```
# the following code obtains the optimal weights and MSE

X = []
y = []
COV_X = []
for three_sample_before, two_sample_before, one_sample_before, curr in zip(data[0:-3:]
    vector1 = np.array([one_sample_before, two_sample_before, three_sample_before])
    vector2 = np.array([curr, one_sample_before, two_sample_before]) # for R
    X.append(vector1)
    y.append(curr)
    COV_X.append(vector2)
X = np.vstack(X)
y = np.array(y)
COV_X = np.vstack(COV_X)


R = np.cov(COV_X.T)
p = y.dot(X)/len(y)
w = np.linalg.inv(R).dot(p)
prediction = X.dot(w)
print("Optimal Weights:",w)
error = y.dot(y)/n_sample - p.dot(w)
print("Optimal MSE:",error)
```

```
Optimal Weights: [-0.68014248 -0.41424331 -0.17878953]
Optimal MSE: 0.3513533155728955
```

In [44]:

```
# here are some helper functions
def quantize(values,BW):
    quant = np.minimum(np.round(np.array(values)*np.power(2.0,BW-1.0))*np.power(2.0,1.(
    return quant

def calc_MSE(x,x_hat):
    #here x is true values, x_hat is prediction
        return np.sum(np.square(np.array(x) - np.array(x_hat)))/n_sample

def evaluate_sqnr(data, dq,mserr):
    SNR = 10*np.log10(np.true_divide(np.sum(np.var(data)),np.sum(np.var(np.array(data)
    return SNR

def predict(wq,xq):
    x_hats = np.matmul(xq,np.transpose(wq))
    mse = calc_MSE(y,x_hats)
    return mse


#first without quantizing:
print("Optimal MSE:",error)
optimal_SNR = 10*np.log10(np.var(y)/error)
print("Optimal SNR (dB):", optimal_SNR)

#now quantize for various bit precisions
precisions = np.arange(1,17,1)

print("Bits \t| MSE \t| SQNR\t|SQNR-SNR|\tWithin .5dB of SNR?")
MSEs = []
sqnrs = []
for BW in precisions:
    wq = quantize(w,BW)
    dq = quantize(X,BW)
    MSE = predict(wq,dq)
    MSEs.append(MSE)
    sqnr = evaluate_sqnr(X,dq,MSE)
    sqnrs.append(sqnr)
    print("{:.4f}\t|{:.4f}\t|{:.3f}\t|{:.4f}|\t{}".format(BW,MSE,sqnr,abs(sqnr-optimal_
```

```
Optimal MSE: 0.3513533155728955
Optimal SNR (dB): 1.698789959975315
Bits     | MSE    | SQNR   |SQNR-SNR|      Within .5dB of SNR?
1.0000   |0.5718 |-1.896 |3.5949|        NO
2.0000   |0.4453 |-0.053 |1.7515|        NO
3.0000   |0.3851 |0.900   |0.7987|        NO
4.0000   |0.3698 |1.208   |0.4906|        YES
5.0000   |0.3654 |1.312   |0.3870|        YES
6.0000   |0.3634 |1.357   |0.3417|        YES
7.0000   |0.3627 |1.375   |0.3239|        YES
8.0000   |0.3625 |1.383   |0.3162|        YES
9.0000   |0.3623 |1.387   |0.3121|        YES
10.0000 |0.3623 |1.389   |0.3101|        YES
11.0000 |0.3622 |1.390   |0.3091|        YES
12.0000 |0.3622 |1.390   |0.3085|        YES
```

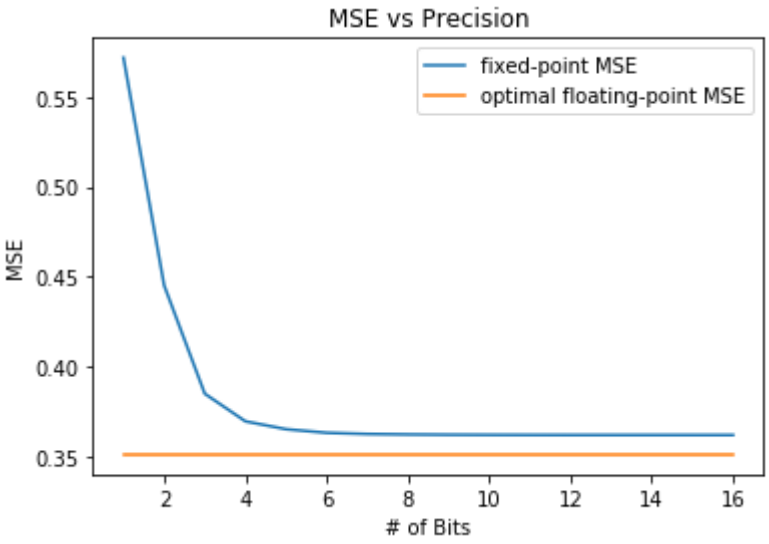
```
13.0000 |0.3622 |1.391  |0.3082|        YES
14.0000 |0.3622 |1.391  |0.3081|        YES
15.0000 |0.3622 |1.391  |0.3081|        YES
16.0000 |0.3622 |1.391  |0.3080|        YES
```

In [28]:

```
# we can also plot the MSE and SQNRs across different bit widths
plt.figure()
plt.plot(precisions, MSEs, label = "fixed-point MSE")
plt.title("MSE vs Precision")
plt.xlabel("# of Bits")
plt.ylabel("MSE")
plt.plot(precisions,[error]*len(precisions),label = "optimal floating-point MSE")
plt.legend()
```

Out[28]:

```
<matplotlib.legend.Legend at 0x17a0a5b5198>
```

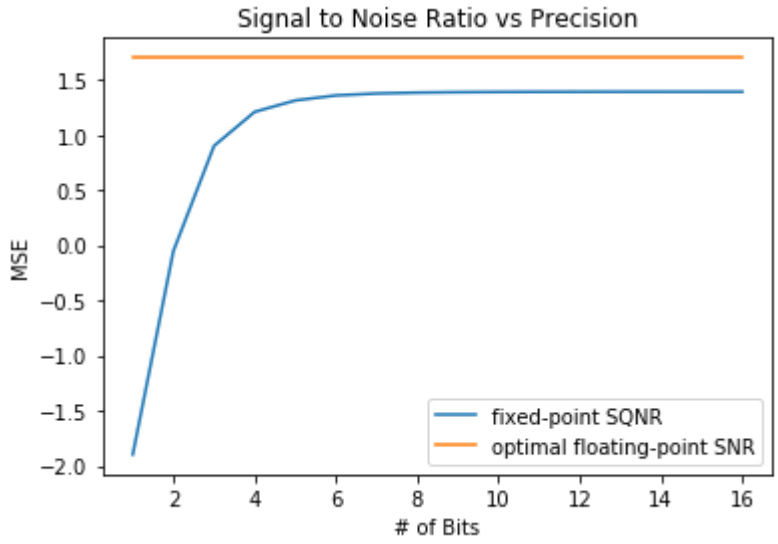


In [47]:

```
plt.figure()
plt.plot(precisions, sqnrs, label = "fixed-point SQNR")
plt.title("Signal to Noise Ratio vs Precision")
plt.xlabel("# of Bits")
plt.ylabel("MSE")
plt.plot(precisions,[optimal_SNR]*len(precisions),label = "optimal floating-point SNR"
plt.legend()
```

Out[47]:

```
<matplotlib.legend.Legend at 0x17a0a48c9e8>
```



In [81]:

```
#verilog confirmation:
xs = np.array([np.array([0.73500254,  0.55186864,  0.19778589]),[-0.69549231, 0.735002
y_dec= [-0.69549231, -0.66702645]
ws_quant = quantize(w,4)*16
for x in xs[:-1]:
    xs_quant = quantize(x,4)
    print(xs_quant*16,ws_quant)
    xs_quant = xs_quant*16
    print(xs_quant.dot(ws_quant)/256,x.dot(w))
```

```
[12.  8.  4.] [-10.  -6.  -2.]
-0.6875 -0.7638763927922381
```

In [86]:

```
x_dptest = np.array([0.73500254,  0.55186864,  0.19778589])
w_dptest = np.array([-0.68014248, -0.41424331, -0.17878953])
print("Floating point dot-product:", x_dptest.dot(w_dptest))
```

```
Floating point dot-product: -0.7638763887944293
```