In [1]:
```
from zero import Circuit
from zero.components import Resistor, Capacitor, Inductor
from zero.analysis import AcNoiseAnalysis, AcSignalAnalysis
import numpy as np
from pyliso import round_to_nearest_value as Eval
from plotting import plotTF, plotTFs
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages       #For saving figures t
figlist = []
#*****************************************************************************
#Setting RC Parameters for figure size and fontsizes
import matplotlib.pylab as pylab
params = {'legend.fontsize': 'xx-large',
          'figure.figsize': (20, 10),
         'axes.labelsize': 'xx-large',
         'axes.titlesize':'xx-large',
         'xtick.labelsize':'xx-large',
         'ytick.labelsize':'xx-large'}
pylab.rcParams.update(params)
#*****************************************************************************
import scipy
```

Sensor conversion slope

Cavity reflection

The reflection trasnfer fuction from a cavity is given by (Eq 3.1 in Black et al. Am. J. Phys., 69, 1 (2001) (https://aapt.scitation.org/doi/pdf/10.1119/1.1286663?class=pdf)):

$$F[\omega] = \frac{r(1-e^{\imath\frac{\omega}{\nu_{FSR}}})}{1-r^2e^{\imath\frac{\omega}{\nu_{FSR}}}}$$

where r is the cavity mirror reflectivity and ν_{FSR} is the cavity free spectral range.

PDH Error signal

From Section 4 in (Black et al. Am. J. Phys., 69, 1 (2001) (https://aapt.scitation.org/doi/pdf/10.1119/1.1286663?class=pdf)), for fast modulation frequencies ($\Omega >> \delta\nu$):

$$\epsilon_{PDH} = 2\sqrt{P_C P_S}\mathcal{I}m(F(\omega)F(\omega+\Omega)^* - F(\omega)^*F(\omega-\Omega))$$

where $P_C = J_0^2(\beta)P_0$ is carrier power, $P_S = J_1^2(\beta)P_0$ is the sideband power, β being modulation index, and $\delta\nu$ is cavity linewidth.

Since $\Omega >> \delta\nu$, the sidebands reflect almost completely giving $F(\omega \pm \Omega) \approx -1$. This gives:

$$\epsilon_{PDH} = -4\sqrt{P_C P_S}\mathcal{I}m(F(\omega))$$

$$\epsilon_{PDH} = -4\sqrt{P_C P_S}\mathcal{I}m\left(\frac{r(-1+r^2e^{-\imath\frac{\omega}{\nu_{FSR}}}+e^{\imath\frac{\omega}{nu_{FSR}}}-r^2)}{1+r^4-2r^2cos(\frac{\omega}{\nu_{FSR}})}\right)$$

$$\epsilon_{PDH} = -\frac{4\sqrt{P_C P_S}r(1-r^2)sin(\frac{\omega}{\nu_{FSR}})}{1+r^4-2r^2cos(\frac{\omega}{\nu_{FSR}})}$$

Now for $\omega = \delta\omega + 2n\pi\nu_{FSR}$ where $\delta\omega$ is the small error from a near by resonance, we can extend sine and cosine terms keeping upto quadratic order:

$$\epsilon_{PDH} = -\frac{4\sqrt{P_C P_S}r(1-r^2)}{(1-r^2)^2+r^2(\frac{\delta\omega}{\nu_{FSR}})^2}\frac{\delta\omega}{\nu_{FSR}}$$

Therefore, trasnfer function of PDH mechanism from error in frequency $\delta f = \frac{\delta\omega}{2\pi}$ to error power (W) signal ϵ_{PDH} is:

$$H_{PDH} = \frac{\epsilon_{PDH}}{\delta f} = -\frac{4\sqrt{P_C P_S}r(1-r^2)}{(1-r^2)^2+r^2(\frac{2\pi\delta f}{\nu_{FSR}})^2}\frac{2\pi}{\nu_{FSR}}$$

$$H_{PDH} = -\frac{2\sqrt{P_C P_S}(1-r^2)\nu_{FSR}}{\pi r(f_p-\imath f)(f_p+\imath f)} \quad \frac{W}{Hz}$$

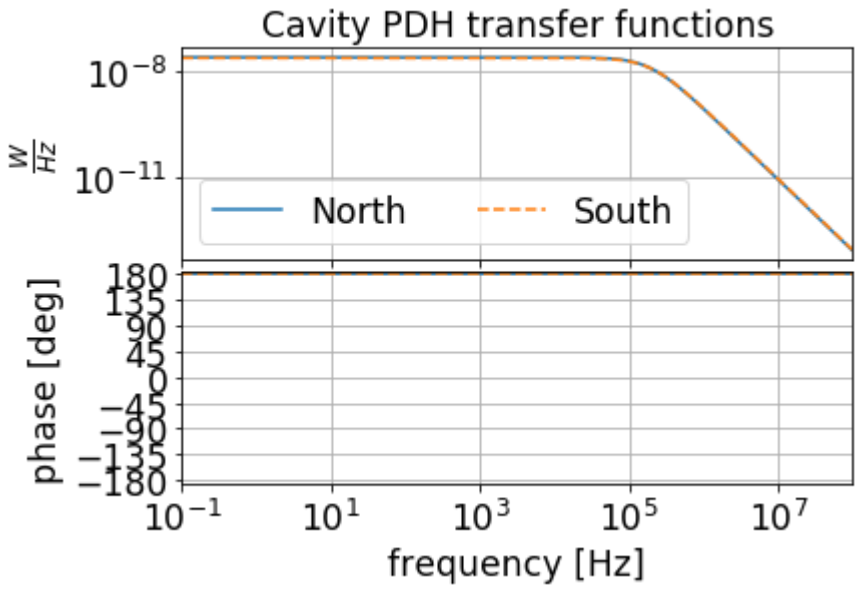
$$H_{PDH} = -\frac{4\sqrt{P_C P_S}}{f_p}\frac{f_p^2}{(f_p-\imath f)(f_p+\imath f)} \quad \frac{W}{Hz}$$

where $f_p = \frac{(1-r^2)\nu_{FSR}}{2\pi r}$ is the cavity pole.

In [2]:
```
ff = np.logspace(-1,8, 800)
P0 = 8e-3           # W, Incident power on cavity
beta = 0.3          # PDH modulation index
Pc = P0*scipy.special.jv(0,beta)**2   # Carrier Power
Ps = P0*scipy.special.jv(1,beta)**2   # Sideband Power

N_T = 297e-6    # North mirror transmission
S_T = 310e-6    # South mirror transmission
N_r = np.sqrt(1-N_T)   # North mirror reflectivity
S_r = np.sqrt(1-S_T)   # South mirror reflectivity

L = 1.45*25.4e-3    # m, Cavity Length
c = scipy.constants.c             # m/s Speed of light
FSR = c/(2*L)
N_fp = N_T*FSR/N_r/2/np.pi   # North Cavity Pole
S_fp = S_T*FSR/S_r/2/np.pi   # South Cavity Pole

print('North Cavity pole:',np.round(N_fp/1e3,2),'kHz')
print('South Cavity pole:',np.round(S_fp/1e3,2),'kHz')

N_H_PDH = -4*np.sqrt(Pc*Ps)*N_fp/((N_fp - 1j*ff)*(N_fp + 1j*ff))
S_H_PDH = -4*np.sqrt(Pc*Ps)*S_fp/((S_fp - 1j*ff)*(S_fp + 1j*ff))

fig = plotTFs(ff, {'North':N_H_PDH, 'South':S_H_PDH})
fig.axes[0].set_title('Cavity PDH transfer functions')
fig.axes[0].set_ylabel(r'$\frac{W}{Hz}$')
```

```
North Cavity pole: 192.41 kHz
South Cavity pole: 200.83 kHz
```

Out[2]:
```
Text(0, 0.5, '$\\frac{W}{Hz}$')
```



RFPD

The RFPDs transimpedance and responsivity converts this into V. So our sensors frequency error to V conversion is:

$$H_{RFPD} = Z_{TI}\mathcal{R} \quad \frac{V}{W}$$

where Z_{TI} is transimpedance gain of RFPD amplifier and $\mathcal{R}$ is the photodiode's responsivity.

Note, we are ignoring any shaping due to RFPD resonance near the peak assuming it is flat enough upto +/- 1 MHz.

In [3]:
```
S_Z_TI = 937     # V/A, RFPD transimpedance
N_Z_TI = 1816    # V/A, RFPD transimpedance
Res = 0.75          # A/W, Photodiode responsivity

N_H_RFPD = N_Z_TI*Res
S_H_RFPD = S_Z_TI*Res
```

Mixer loss and low pass filter

In reality the error signal is on the modulation frequency and gets demodulated inside the FSS. The mixer used is JMS-1H (https://www.minicircuits.com/pdfs/JMS-1H.pdf). The mixer will have some loss A_m: and this is followed by elliptical filter:

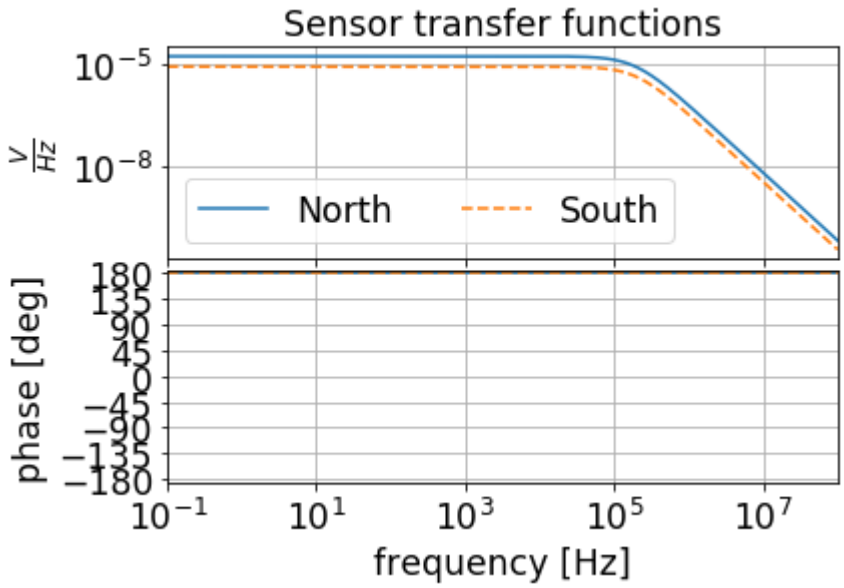
$$H_{Mixer} = A_m$$

Total Sensor Transfer Function

$$H_{Sensor} = H_{PDH}H_{RFPD}H_{Mixer,Filter} \quad \frac{V}{Hz}$$

In [4]:
```
Am = 0.506991    # Equivalent to 5.9 dB loss
H_Mixer = Am

N_H_Sensor = N_H_PDH*N_H_RFPD*H_Mixer
S_H_Sensor = S_H_PDH*S_H_RFPD*H_Mixer

fig = plotTFs(ff, {'North':N_H_Sensor, 'South':S_H_Sensor})
fig.axes[0].set_title('Sensor transfer functions')
fig.axes[0].set_ylabel(r'$\frac{V}{Hz}$')
```

Out[4]:
```
Text(0, 0.5, '$\\frac{V}{Hz}$')
```



PZT actuation slope

PZT actuation slope is directly marked on the laser as 1 MHz/V. So

$$H_{PZT} = 10^6 \ \frac{Hz}{V}$$

EOM actuation slope

EOM actuates on the phase of the laser, not the frequency directly. [New Focus 4004 (https://www.newport.com/medias/sys_master/images/images/h65/hcc/8797007839262/400X-and-406X-User-Manual-Rev-J.pdf)](https://www.newport.com/medias/sys_master/images/images/h65/hcc/8797007839262/400X-and-406X-User-Manual-Rev-J.pdf) has a modulation slope of $m_s = 15mrad/V$. The rate of change of the phase acts as frequency change by the EOM:

$$\delta f(t) = \frac{1}{2\pi}\frac{d\Delta\phi}{dt} = \frac{m_s}{2\pi}\frac{dV_{EOM}}{dt}$$

Therefore, in fourier domain,

$$\delta\tilde{f}(f) = \frac{m_s}{2\pi}(-\imath 2\pi f)\tilde{V_{EOM}}(f)$$

Therefore, the actuation slope for the EOM is:

$$H_{EOM} = -\imath f m_s \ \frac{Hz}{V}$$

Plant Circuit

In Plant circuit, the frequency is stored as symbolic voltage V_S where the conversion is:

$$\eta_{HztoVs} = 1e6$$

i.e. 1 MHz is stored as 1 V_S

In [5]:
```
ms = 15e-3          # rad/V EOM modulation slope
HztoVs = 1e6        # Hz/Vs, 1 MHz is stored as 1 Vs

H_PZT = 1e6                                 # Hz/V
H_PZTVs = H_PZT/HztoVs                      # Vs/V

H_EOM = -1j*ff*ms                           # Hz/V
H_EOMVs = H_EOM/HztoVs                      # Vs/V

N_H_SensorVs = N_H_Sensor*HztoVs            # V/Vs
S_H_SensorVs = S_H_Sensor*HztoVs            # V/Vs
```

Mapping requried TF to plant circuit values

Gain on PZT path (including sensor transfer function without poles of PDH):

$$-\frac{R3}{R1} = H_{PZT,Vs} H_{RFPD} H_{Mixer} \frac{-4\sqrt{P_C P_S}}{f_P} \eta_{HzToVs}$$

$$\frac{R3}{R1} = H_{PZT} H_{RFPD} H_{Mixer} \frac{4\sqrt{P_C P_S}}{f_P}$$

On the EOM path (including sensor transfer function without poles of PDH):

$$(1 + \frac{R3}{R1})(\frac{\imath f}{\frac{1}{2\pi R_4 C_2} + \imath f}) = H_{EOM,Vs} H_{RFPD} H_{Mixer} \frac{-4\sqrt{P_C P_S}}{f_P} \eta_{HzToVs}$$

$$(1 + \frac{R3}{R1})(\frac{\imath f}{\frac{1}{2\pi R_4 C_2} + \imath f}) = -\imath f m_s H_{RFPD} H_{Mixer} \frac{-4\sqrt{P_C P_S}}{f_P}$$

Approximately for $f << \frac{1}{2\pi R_4 C_2}$:

$$(1 + \frac{R3}{R1}) 2\pi R_4 C_2 = \frac{H_{RFPD} H_{Mixer} m_s 4\sqrt{P_C P_S}}{f_P}$$

In [6]:
```
R3_over_R1S = H_PZT * S_H_RFPD * H_Mixer * 4 * np.sqrt(Pc*Ps)/S_fp
R4C2S = (ms * N_H_RFPD * H_Mixer * 4 * np.sqrt(Pc*Ps)/S_fp) /(1+R3_over_R1S) / 1
R3_over_R1N = H_PZT * N_H_RFPD * H_Mixer * 4 * np.sqrt(Pc*Ps)/N_fp
R4C2N = (ms * S_H_RFPD * H_Mixer * 4 * np.sqrt(Pc*Ps)/N_fp) /(1+R3_over_R1N) / 3
print('South: R3/R1:', np.round(R3_over_R1S,2),
      'R4C2:', np.round(R4C2S*1e9, 2), 'ns',
      'High Pass 3-dB cutoff:',
      np.round(1/2/np.pi/R4C2S/1e6, 2), 'MHz')
print('North: R3/R1:', np.round(R3_over_R1N,2),
      'R4C2:', np.round(R4C2N*1e9, 2), 'ns',
      'High Pass 3-dB cutoff:',
      np.round(1/2/np.pi/R4C2N/1e6, 2), 'MHz')
```

```
South: R3/R1: 8.23 R4C2: 2.12 ns High Pass 3-dB cutoff: 75.13 MHz
North: R3/R1: 16.65 R4C2: 2.27 ns High Pass 3-dB cutoff: 70.09 MHz
```

In [7]:
```
R1N = Eval(1000, 'E12')
R3N = Eval(R1N*R3_over_R1N, 'E12')
R4N = Eval(10, 'E12')
C2N = Eval(R4C2N/R4N, 'E12')
R1S = Eval(1000, 'E12')
R3S = Eval(R1S*R3_over_R1S, 'E12')
R4S = Eval(10, 'E12')
C2S = Eval(R4C2S/R4S, 'E12')

print('South: R1:', R1S,'R3:', R3S, 'R4:', R4S, 'C2:', C2S)
print('North: R1:', R1N,'R3:', R3N, 'R4:', R4N, 'C2:', C2N)
```

```
South: R1: 1000.0 R3: 8250.0 R4: 10.0 C2: 2.2e-10
North: R1: 1000.0 R3: 17800.0 R4: 10.0 C2: 2.2e-10
```

In [8]:
```
R5C5S = 1/2/np.pi/S_fp
R5C5N = 1/2/np.pi/N_fp
print('South: R5C5:', np.round(R5C5S*1e9, 2), 'ns')
print('North: R5C5:', np.round(R5C5N*1e9, 2), 'ns')

R5S = Eval(100, 'E12')
C5S = Eval(R5C5S/R5S, 'E12')
R6S = R5S
C6S = C5S
R5N = Eval(100, 'E12')
C5N = Eval(R5C5N/R5N, 'E12')
R6N = R5N
C6N = C5N

print('South: R5:', R5S,'C5:', C5S, 'R6:', R6S, 'C6:', C6S)
print('North: R5:', R5N,'C5:', C5N, 'R6:', R6N, 'C6:', C6N)
```

```
South: R5C5: 792.47 ns
North: R5C5: 827.16 ns
South: R5: 100.0 C5: 8.3e-09 R6: 100.0 C6: 8.3e-09
North: R5: 100.0 C5: 8.3e-09 R6: 100.0 C6: 8.3e-09
```

In [9]:
```
plantS = Circuit()
plantN = Circuit()
plantS.add_capacitor(name='PZTC', value='10n', node1='PZTnIN', node2='gnd')
plantS.add_capacitor(name='EOMC', value='20p', node1='EOMnIN', node2='gnd')
plantN.add_capacitor(name='PZTC', value='10n', node1='PZTnIN', node2='gnd')
plantN.add_capacitor(name='EOMC', value='20p', node1='EOMnIN', node2='gnd')

plantS.add_library_opamp(name='Buffer', model='ad829', node1='PZTnIN',
                         node2='PZTnINa', node3='PZTnINa')
plantS.add_resistor(name='R1', value=R1S, node1='PZTnINa', node2='n2')
plantS.add_resistor(name='R3', value=R3S, node1='n2', node2='n3')
plantS.add_capacitor(name='C2', value=C2S, node1='EOMnIN', node2='n1')
plantS.add_resistor(name='R4', value=R4S, node1='n1', node2='gnd')
plantS.add_library_opamp(name='Adder', model='ad829', node1='n1',
                         node2='n2', node3='n3')

plantS.add_resistor(name='R5', value=R5S, node1='n3', node2='n4')
plantS.add_resistor(name='R6', value=R6S, node1='n4', node2='n5')
plantS.add_capacitor(name='C5', value=C5S, node1='n4', node2='nOut')
plantS.add_capacitor(name='C6', value=C6S, node1='n5', node2='gnd')
plantS.add_resistor(name='R7', value=100, node1='n6', node2='nOut')
plantS.add_library_opamp(name='CavityPole', model='ad829', node1='n5',
                         node2='n6', node3='nOut')

plantN.add_library_opamp(name='Buffer', model='ad829', node1='PZTnIN',
                         node2='PZTnINa', node3='PZTnINa')
plantN.add_resistor(name='R1', value=R1N, node1='PZTnINa', node2='n2')
plantN.add_resistor(name='R3', value=R3N, node1='n2', node2='n3')
plantN.add_capacitor(name='C2', value=C2N, node1='EOMnIN', node2='n1')
plantN.add_resistor(name='R4', value=R4N, node1='n1', node2='gnd')
plantN.add_library_opamp(name='Adder', model='ad829', node1='n1',
                         node2='n2', node3='n3')

plantN.add_resistor(name='R5', value=R5N, node1='n3', node2='n4')
plantN.add_resistor(name='R6', value=R6N, node1='n4', node2='n5')
plantN.add_capacitor(name='C5', value=C5N, node1='n4', node2='nOut')
plantN.add_capacitor(name='C6', value=C6N, node1='n5', node2='gnd')
plantN.add_resistor(name='R7', value=100, node1='n6', node2='nOut')
plantN.add_library_opamp(name='CavityPole', model='ad829', node1='n5',
                         node2='n6', node3='nOut')
```

In [10]:
```
sigAnalS = AcSignalAnalysis(circuit=plantS)
TFfromPZTS = sigAnalS.calculate(frequencies=ff,
                                input_type='voltage',
                                node='PZTnIN')
TFfromEOMS = sigAnalS.calculate(frequencies=ff,
                                input_type='voltage',
                                node='EOMnIN')

sigAnalN = AcSignalAnalysis(circuit=plantN)
TFfromPZTN = sigAnalN.calculate(frequencies=ff,
                                input_type='voltage',
                                node='PZTnIN')
TFfromEOMN = sigAnalN.calculate(frequencies=ff,
                                input_type='voltage',
                                node='EOMnIN')

noiseAnalS = AcNoiseAnalysis(circuit=plantS)
noiseS = noiseAnalS.calculate(frequencies=ff,
                              input_type="voltage",
                              node="PZTnIN",
                              sink="nOut",
                              incoherent_sum=True)

noiseAnalN = AcNoiseAnalysis(circuit=plantN)
noiseN = noiseAnalN.calculate(frequencies=ff,
                              input_type="voltage",
                              node="PZTnIN",
                              sink="nOut",
                              incoherent_sum=True)
```

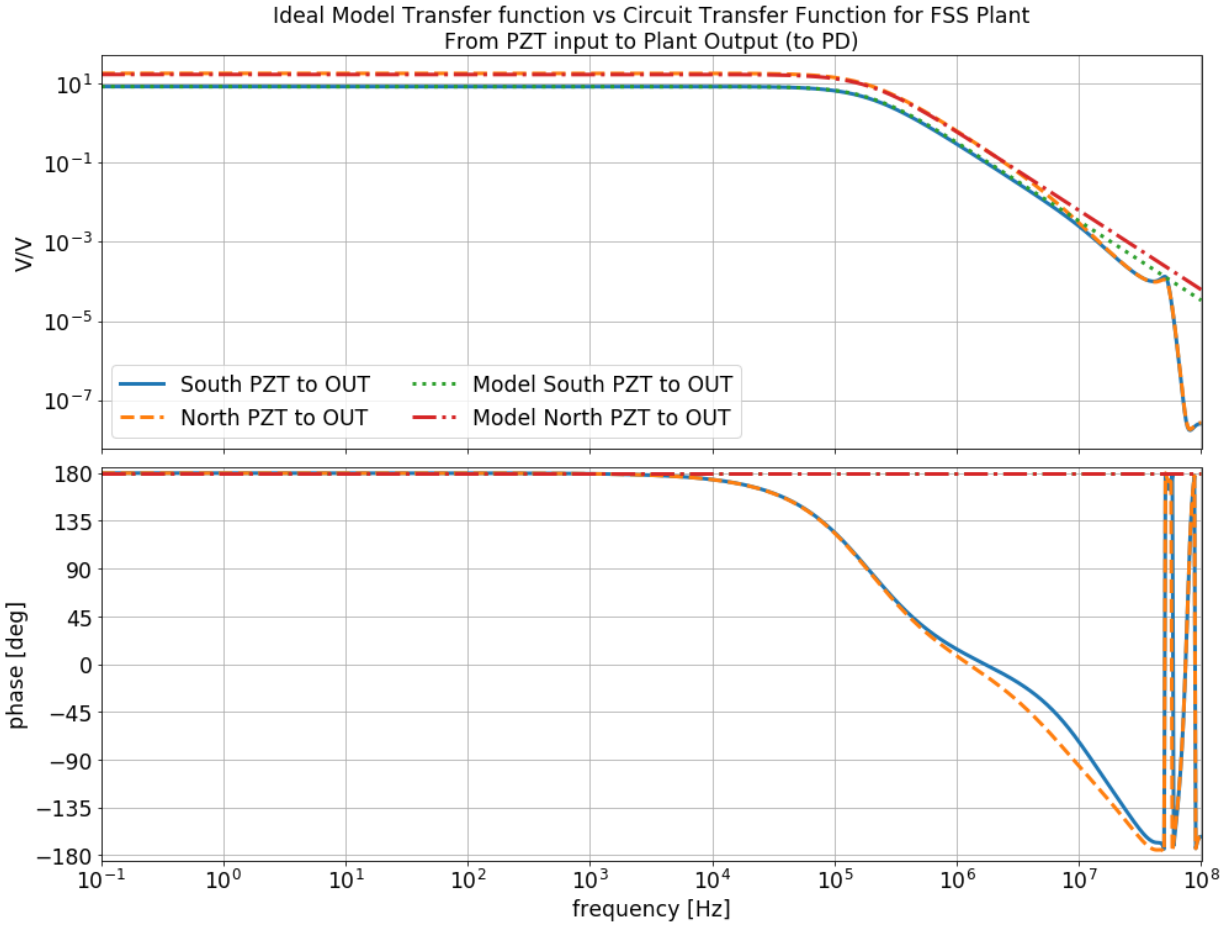
```
WARNING: assuming default input impedance of 50 (zero.analysis.ac.noise)
WARNING: assuming default input impedance of 50 (zero.analysis.ac.noise)
```

In [11]:
```
TFDict1 = {}
TFDict2 = {}
TFDict1['South PZT to OUT'] = TFfromPZTS.get_response("PZTnIn", "nOut").complex_
TFDict2['South EOM to OUT'] = TFfromEOMS.get_response("EOMnIn", "nOut").complex_
TFDict1['North PZT to OUT'] = TFfromPZTN.get_response("PZTnIn", "nOut").complex_
TFDict2['North EOM to OUT'] = TFfromEOMN.get_response("EOMnIn", "nOut").complex_
TFDict1['Model South PZT to OUT'] = H_PZTVs*S_H_SensorVs
TFDict2['Model South EOM to OUT'] = H_EOMVs*S_H_SensorVs
TFDict1['Model North PZT to OUT'] = H_PZTVs*N_H_SensorVs
TFDict2['Model North EOM to OUT'] = H_EOMVs*N_H_SensorVs
```

In [12]:
```
np.abs(TFDict2['South EOM to OUT'][0])/np.abs(TFDict2['Model South EOM to OUT'][
```

Out[12]:
```
1.0354436370568132
```

In [13]:
```
fig = plotTFs(ff, TFDict1, figsize=[16,12], lw=3)
fig.axes[0].set_ylabel('V/V')
fig.axes[0].set_title('Ideal Model Transfer function vs Circuit Transfer Functic
                      '\nFrom PZT input to Plant Output (to PD)')
fig.axes[1].set_xticks(np.logspace(-1,8,10))
figlist = [fig];
```



In [14]:
```
fig = plotTFs(ff, TFDict2, figsize=[16,12], lw=3)
fig.axes[0].set_ylabel('V/V')
fig.axes[0].set_title('Ideal Model Transfer function vs Circuit Transfer Functic
                      '\nFrom EOM input to Plant Output (to PD)')
fig.axes[1].set_xticks(np.logspace(-1,8,10))
figlist += [fig];
```