Fast Digital Servo:
ADC Converter Module
T2400199-v1

Initialization

In[ ]:=
```
Needs["Controls`LinearControl`"]
```

In[ ]:=
```
$TextStyle = {FontFamily → "Helvetica", FontSize → 13};
```

In[ ]:=
```
plotopt = Sequence @@ {GridLines → Automatic, Frame → True,
    FrameStyle → Thickness[0.0025], BaseStyle → {FontSize → 12}};
```

In[ ]:=
```
plotoptn[n_Integer ? (# > 0 ∧ # < 9 &)] :=
  Sequence @@ {GridLines → Automatic, Frame → True, FrameStyle → Thickness[0.0025],
    PlotStyle → Take[{Black, Gray, Darker[Yellow], Purple, Brown,
       Darker[Green], Blue, Red}, -n], BaseStyle → {FontSize → 12}};
plotoptn[n_Integer ? (# ≤ 0 ∨ # ≥ 9 &)] := plotopt
```

In[ ]:=
```
mylegend[labels_List, pos_ : Right] :=
 {Placed[LineLegend[labels, LabelStyle → {FontSize → 11}, LegendMargins → 2,
    LegendFunction → (Framed[#, Background → White] &)], pos]}
```

Parallel and Serial Impedance

In[ ]:=
```
par[r1_, r2_] := 1/(1/r1 + 1/r2)
par[r1_, r2_, r3_] := 1/(1/r1 + 1/r2 + 1/r3)
par[r1_, r2_, r3_, r4__] := 1/(1/r1 + 1/r2 + 1/r3 + Plus @@ (1/List[r4]))
ser[r1_, r2_] := r1 + r2
ser[r1_, r2_, r3_] := r1 + r2 + r3
ser[r1_, r2_, r3_, r4__] := r1 + r2 + r3 + Plus @@ List[r4]
```

In[ ]:=
```
fourkt = {FourKT → 1.62*^-20};(* V^2/Hz/Ohm; room temperature 20C *)
```

Pole/Zero

In[ ]:=
```
pole[f_, p_] := 1/(1 + ⅈ f / p)
zero[f_, p_] := 1 + ⅈ f / p
pole[f_, p_, Q_] := 1/(1 + ⅈ (1/Q) (f/p) - (f / p)^2)
zero[f_, p_, Q_] := 1 + ⅈ (1/Q) (f/p) - (f / p)^2
```

Parallel Capacitors

Transfer Function of an OpAmp

This function computes the transfer function of an idealized OpAmp circuit
g: +1 for non-inverting configuration or -1 for inverting configuration, 0 for differential configuration
z2: Impedance in feedback path [Ohm]
z1: Impedance of input path (inverting) or impedance to ground (non-inverting) [Ohm]

In[ ]:=
```
OpAmp[g_, z1_, z2_] :=
 Which[g > 0, 1 + z2/z1, g < 0, -z2/z1, True, z2/z1]
```

Noise of an OpAmp

This function computes the equivalent input noise of an OpAmp circuit
g: +1 for non-inverting configuration or -1 for inverting configuration, 0 for differential configuration

z1: Impedance of input path (inverting) or impedance to ground (non-inverting) [Ohm]
z2: Impedance over feedback path [Ohm]
en: voltage noise [Volt]
in: current noise [Ampere]

In[∘]:=
```
OpAmpNoise[g_, z1_, z2_, en_, in_] :=
 Which[g > 0, If[z1 == Infinity, Sqrt[en^2 + FourKT Abs[z2] + (in Abs[z2])^2],
    Sqrt[en^2 + FourKT Abs[par[z1, z2]] + (in Abs[par[z1, z2]])^2]],
  g < 0, Sqrt[(Abs[1 + z1/z2]^2 en^2 + Abs[z1]^2 (in^2 + Abs[FourKT/z1] + Abs[FourKT/z2]))],
  True, Sqrt[(Abs[1 + z1/z2]^2 en^2 + 2 Abs[z1]^2 (in^2 + Abs[FourKT/z1] + Abs[FourKT/z2]))]]
```

Flicker Noise: The variable $Flicker determines if flicker noise is added or not. It can also be explicitly specified with the option Flicker.

In[∘]:=
```
$Flicker = True;
Clear[OpAmpNoiseFlicker];
Options[OpAmpNoiseFlicker] = {Flicker :> $Flicker};
OpAmpNoiseFlicker[f_, fknee_, opts___] :=
 If[Flicker /. {opts} /. Options[OpAmpNoiseFlicker], Sqrt[fknee/f + 1], 1]
OpAmpNoiseFlicker[f_, fknee_, floor_, opts___] := floor OpAmpNoiseFlicker[f, fknee, opts]
```

Series Product of OpAmps

Computes the transfer function of several OpAmps circuits in series.

In[∘]:=
```
OpAmpProduct[t_, m_] := Product[t[i], {i, m}]
```

Computes the equivalent input noise of several OpAmps circuits in series.

In[∘]:=
```
NoiseSum[prev_, {t_, n_}] := Sqrt[prev^2 + n^2] Abs[t]
OpAmpNoiseProduct[t_, n_, m_] := Fold[NoiseSum, 0, Table[{t[i], n[i]}, {i, m}]] / Abs[OpAmpProduct[t, m]]
OpAmpNoiseProductOut[t_, n_, m_] := Fold[NoiseSum, 0, Table[{t[i], n[i]}, {i, m}]]
```

Spectrum Math

Propagate noise spectrum

```
In[•]:= SpecProp[prev_, t_] := {#[[1]], Abs[t /. s → 2. π #[[1]]] #[[2]]} & /@ prev
        SpecProp[noise_, t_, m_] := FoldList[SpecProp, noise, Table[t[i], {i, m}]]
```

RMS of spectrum

```
In[•]:= Clear[SpecRMS];
        SpecRMS[l_List ? (MatrixQ[#, NumberQ] &)] := Block[{i, sqr = 0},
          For[i = 1, i < Length[l], ++i,
           sqr += (l[[i + 1, 1]] - l[[i, 1]]) ((l[[i, 2]] + l[[i + 1, 2]])/2)^2];
          Sqrt[sqr]]
```

Integrated RMS spectrum

```
In[•]:= Clear[RMSSpec];
        RMSSpec[l_List ? (MatrixQ[#, NumberQ] &), dir_ : (-1)] := Block[{i, sqr = 0, r = N[l]},
          If[dir ≥ 0,
           For[i = 2, i ≤ Length[l], ++i,
            r[[i, 2]] = Sqrt[r[[i - 1, 2]]^2 + r[[i, 2]]^2 (r[[i, 1]] - r[[i - 1, 1]])]],
           For[i = Length[l] - 2, i ≥ 1, --i,
            r[[i, 2]] = Sqrt[r[[i + 1, 2]]^2 + r[[i, 2]]^2 (r[[i + 1, 1]] - r[[i, 1]])]]];
          r]
```

OpAmp Parameters

```
In[•]:= Clear[AD829, OP27]
        AD829[f_] := {s → 2 π ⅈ f, en → enAD829, in → inAD829,
            enfloor → enfloorAD829, infloor → infloorAD829} //.
           {enAD829 → OpAmpNoiseFlicker[f, ekneeAD829, enfloorAD829],
            inAD829 → OpAmpNoiseFlicker[f, ikneeAD829, infloorAD829],
            ekneeAD829 → 50, ikneeAD829 → 100, (*guess*)
            enfloorAD829 → 1.7*^-9, infloorAD829 → 1.5*^-12};
        OP27[f_] := {s → 2 π ⅈ f, en → enOP27, in → inOP27,
            enfloor → enfloorOP27, infloor → infloorOP27} //.
           {enOP27 → OpAmpNoiseFlicker[f, ekneeOP27, enfloorOP27],
            inOP27 → OpAmpNoiseFlicker[f, ikneeOP27, infloorOP27],
            ekneeOP27 → 2.7, ikneeOP27 → 140,
            enfloorOP27 → 3.0*^-9, infloorOP27 → 0.4*^-12};
        LT1028[f_] := {s → 2 π ⅈ f, en → enLT1028, in → inLT1028,
            enfloor → enfloorLT1028, infloor → infloorLT1028} //.
```

```
    {enLT1028 → OpAmpNoiseFlicker[f, ekneeLT1028, enfloorLT1028],
     inLT1028 → OpAmpNoiseFlicker[f, ikneeLT1028, infloorLT1028],
     ekneeLT1028 → 3.5, ikneeLT1028 → 250,
     enfloorLT1028 → 0.85*^-9, infloorLT1028 → 1*^-12};
LT1128[f_] := {s → 2 π ⅈ f, en → enLT1128, in → inLT1128,
     enfloor → enfloorLT1128, infloor → infloorLT1128} //.
    {enLT1128 → OpAmpNoiseFlicker[f, ekneeLT1128, enfloorLT1128],
     inLT1128 → OpAmpNoiseFlicker[f, ikneeLT1128, infloorLT1128],
     ekneeLT1128 → 3.5, ikneeLT1128 → 250,
     enfloorLT1128 → 0.85*^-9, infloorLT1128 → 1*^-12};
LT1007[f_] := {s → 2 π ⅈ f, en → enLT1007, in → inLT1007,
     enfloor → enfloorLT1007, infloor → infloorLT1007} //.
    {enLT1007 → OpAmpNoiseFlicker[f, ekneeLT1007, enfloorLT1007],
     inLT1007 → OpAmpNoiseFlicker[f, ikneeLT1007, infloorLT1007],
     ekneeLT1007 → 2.0, ikneeLT1007 → 120,
     enfloorLT1007 → 2.5*^-9, infloorLT1007 → 0.4*^-12};
LT1037[f_] := {s → 2 π ⅈ f, en → enLT1037, in → inLT1037,
     enfloor → enfloorLT1037, infloor → infloorLT1037} //.
    {enLT1037 → OpAmpNoiseFlicker[f, ekneeLT1037, enfloorLT1037],
     inLT1037 → OpAmpNoiseFlicker[f, ikneeLT1037, infloorLT1037],
     ekneeLT1037 → 2.0, ikneeLT1037 → 120,
     enfloorLT1037 → 2.5*^-9, infloorLT1037 → 0.4*^-12};
AD797[f_] := {s → 2 π ⅈ f, en → enAD797, in → inAD797,
     enfloor → enfloorAD797, infloor → infloorAD797} //.
    {enAD797 → OpAmpNoiseFlicker[f, ekneeAD797, enfloorAD797],
     inAD797 → OpAmpNoiseFlicker[f, ikneeAD797, infloorAD797],
     ekneeAD797 → 50, ikneeAD797 → 100, (*guess*)
     enfloorAD797 → 0.9*^-9, infloorAD797 → 2*^-12};
LT1012[f_] := {s → 2 π ⅈ f, en → enLT1012, in → inLT1012,
     enfloor → enfloorLT1012, infloor → infloorLT1012} //.
    {enLT1012 → OpAmpNoiseFlicker[f, ekneeLT1012, enfloorLT1012],
     inLT1012 → OpAmpNoiseFlicker[f, ikneeLT1012, infloorLT1012],
     ekneeLT1012 → 2.5, ikneeLT1012 → 120, (*guess*)
     enfloorLT1012 → 14*^-9, infloorLT1012 → 6*^-15};
LT1792[f_] := {s → 2 π ⅈ f, en → enLT1792, in → inLT1792,
     enfloor → enfloorLT1792, infloor → infloorLT1792} //.
    {enLT1792 → OpAmpNoiseFlicker[f, ekneeLT1792, enfloorLT1792],
     inLT1792 → OpAmpNoiseFlicker[f, ikneeLT1792, infloorLT1792],
     ekneeLT1792 → 30, ikneeLT1792 → 0, (*guess*)
     enfloorLT1792 → 4.2*^-9, infloorLT1792 → 10*^-15};
ADA4625[f_] := {s → 2 π ⅈ f, en → enADA4625, in → inADA4625,
     enfloor → enfloorADA4625, infloor → infloorADA4625} //.
    {enADA4625 → OpAmpNoiseFlicker[f, ekneeADA4625, enfloorADA4625],
     inADA4625 → OpAmpNoiseFlicker[f, ikneeADA4625, infloorADA4625],
     ekneeADA4625 → 15, ikneeADA4625 → 0, (*guess*)
     enfloorADA4625 → 3.3*^-9, infloorADA4625 → 4.5*^-15};
ADA4522[f_] := {s → 2 π ⅈ f, en → enADA4522, in → inADA4522,
```

```
     enfloor → enfloorADA4522, infloor → infloorADA4522} //.
    {enADA4522 → OpAmpNoiseFlicker[f, ekneeADA4522, enfloorADA4522],
     inADA4522 → OpAmpNoiseFlicker[f, ikneeADA4522, infloorADA4522],
     ekneeADA4522 → 0, ikneeADA4522 → 0, (*chopper*)
     enfloorADA4522 → 7*^-9, infloorADA4522 → 1.2*^-12};
ADA4898[f_] := {s → 2 π ⅈ f, en → enADA4898, in → inADA4898,
     enfloor → enfloorADA4898, infloor → infloorADA4898} //.
    {enADA4898 → OpAmpNoiseFlicker[f, ekneeADA4898, enfloorADA4898],
     inADA4898 → OpAmpNoiseFlicker[f, ikneeADA4898, infloorADA4898],
     ekneeADA4898 → 10, ikneeADA4898 → 20,
     enfloorADA4898 → 0.9*^-9, infloorADA4898 → 2.4*^-12};
PA98A[f_] := {s → 2 π ⅈ f, en → enPA98A, in → inPA98A,
     enfloor → enfloorPA98A, infloor → infloorPA98A} //.
    {enPA98A → OpAmpNoiseFlicker[f, ekneePA98A, enfloorPA98A],
     inPA98A → OpAmpNoiseFlicker[f, ikneePA98A, infloorPA98A],
     ekneePA98A → 100(*guess*), ikneePA98A → 120, (*guess*)
     enfloorPA98A → 4*^-9, infloorPA98A → 1*^-12 (*guess*)};
```

ADA4898

ADR4540

ADC Whitening and Gain

Parameters

```
In[•]:= adcgainwhite = {g12 → False, g6 → False, g3 → False, w1 → False, w2 → False};
        impl[1] = impl[2] = impl[3] = impl[4] = impl[5] = impl[6] = impl[7] = ADA4898[f];
        (*ADA4898[f] AD829[f]*)
```

First/Second Stage: Instrumentation Amplifier

```
In[•]:= z1[1] := If[g12, 665/2, ∞]
        z2[1] := If[g12, par[1000, 1/(s 10*^-12)], 50]
        opamp[1] := OpAmp[1, z1[1], z2[1]]
        opampnoise[1] := √2 OpAmpNoise[1, z1[1], z2[1], en, in]
          (* √2 because there are 2 opamps: one for each leg *)
```

In[∘]:=
```
z1[2] := 1000
z2[2] := par[1000, 1/(s 10*^-12)]
opamp[2] := OpAmp[0, z1[2], z2[2]]
opampnoise[2] := OpAmpNoise[0, z1[2], z2[2], en, in]
```

Third Stage: 6dB Gain

In[∘]:=
```
z1[3] := 2000
z2[3] := par[2000, 1/(s 2.7*^-12)]
opamp[3] := If[g6, OpAmp[1, z1[3], z2[3]], 1]
opampnoise[3] := If[g6, OpAmpNoise[1, z1[3], z2[3], en, in], 0]
```

Forth Stage: 3dB Gain

In[∘]:=
```
z1[4] := 2000
z2[4] := par[825, 1/(s 2.7*^-12)]
opamp[4] := If[g3, OpAmp[1, z1[4], z2[4]], 1]
opampnoise[4] := If[g3, OpAmpNoise[1, z1[4], z2[4], en, in], 0]
```

Fifth Stage: Whitening

In[∘]:=
```
z1A[5] := 1580 + 1/(s 10*^-6)
z2A[5] := par[15 000, 1/(s 33*^-12)]
z1B[5] := ∞
z2B[5] := 50
opamp[5] := If[w1, OpAmp[1, z1A[5], z2A[5]], OpAmp[1, z1B[5], z2B[5]]]
opampnoise[5] :=
 If[w1, OpAmpNoise[1, z1A[5], z2A[5], en, in], OpAmpNoise[1, z1B[5], z2B[5], en, in]]
```

Sixth Stage: Whitening

In[∘]:=
```
z1A[6] := 1580 + 1/(s 10*^-6)
z2A[6] := par[15 000, 1/(s 33*^-12)]
z1B[6] := ∞
z2B[6] := 50
opamp[6] := If[w2, OpAmp[1, z1A[6], z2A[6]], OpAmp[1, z1B[6], z2B[6]]]
opampnoise[6] :=
 If[w2, OpAmpNoise[1, z1A[6], z2A[6], en, in], OpAmpNoise[1, z1B[6], z2B[6], en, in]]
```

Seventh Stage: Differential Driver

In[•]:=
```
z1[7] := 2000
z2[7] := par[2000, 1/(s 10*^-12)]
opamp[7] := 2 OpAmp[-1, z1[7], z2[7]] (* factor of 2 due to diff driver *)
opampnoise[7] :=
 Sqrt[2]/2 OpAmpNoise[-1, z1[7], z2[7], en, in] (* factor of Sqrt[2]/2 due to diff driver *)
```

Input Referred Noise

In[•]:=
```
OpAmpNoiseProduct[opamp, opampnoise, 7] /. adcgainwhite //. ADA4898[f] /. fourkt /.
  f → 1000.
OpAmpNoiseProduct[opamp, opampnoise, 7] /. adcgainwhite //. AD829[f] /. fourkt /. f → 1000.
```

Out[•]=

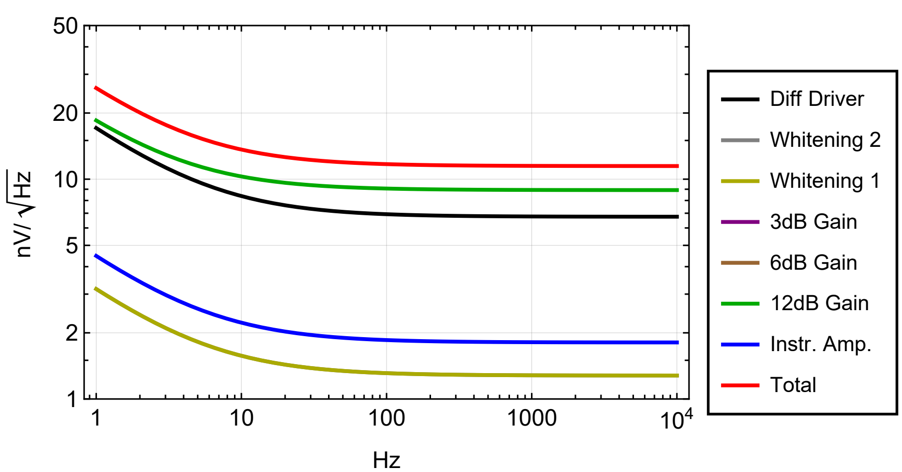
1.1497×10^{-8}

Out[•]=

1.18621×10^{-8}

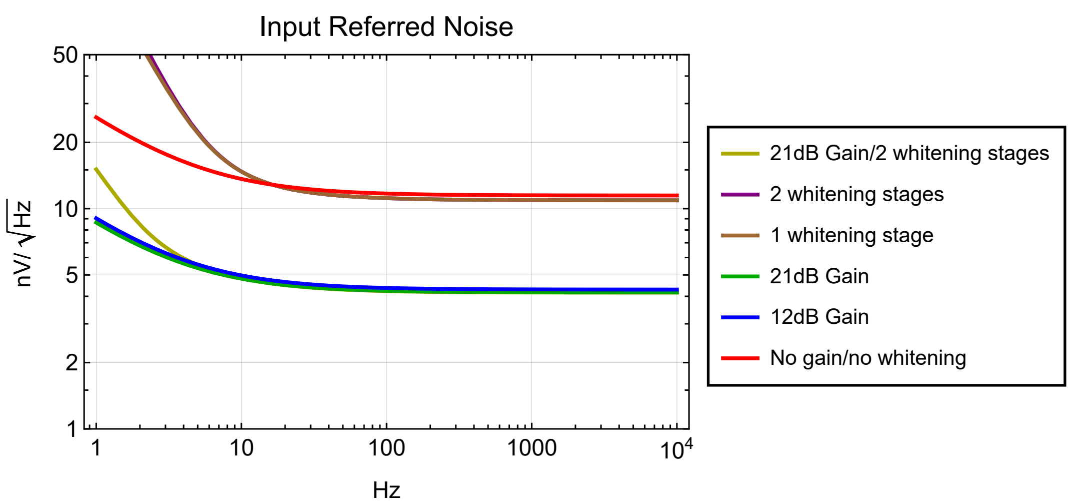
In[•]:=
```
LogLogPlot[Evaluate[
  1*^9 {opampnoise[7], opampnoise[6], opampnoise[5], opampnoise[4], opampnoise[3],
      opampnoise[2], opampnoise[1], OpAmpNoiseProduct[opamp, opampnoise, 7]} /.
    adcgainwhite //. ADA4898[f] /. fourkt], {f, 1, 10*^3},
 PlotRange → {1, 50}, FrameLabel → {"Hz", "nV/ Sqrt[Hz]"},
 Evaluate[plotoptn[8]],
 PlotLegends → mylegend[{"Diff Driver", "Whitening 2", "Whitening 1",
    "3dB Gain", "6dB Gain", "12dB Gain", "Instr. Amp.", "Total"}]]
```

Out[•]=



In[ ]:=
```
LogLogPlot[Evaluate[1*^9 {
      OpAmpNoiseProduct[opamp, opampnoise, 7] /.
        {g12 → True, g6 → True, g3 → True, w1 → True, w2 → True},
      OpAmpNoiseProduct[opamp, opampnoise, 7] /.
        {g12 → False, g6 → False, g3 → False, w1 → True, w2 → True},
      OpAmpNoiseProduct[opamp, opampnoise, 7] /.
        {g12 → False, g6 → False, g3 → False, w1 → True, w2 → False},
      OpAmpNoiseProduct[opamp, opampnoise, 7] /.
        {g12 → True, g6 → True, g3 → True, w1 → False, w2 → False},
      OpAmpNoiseProduct[opamp, opampnoise, 7] /.
        {g12 → True, g6 → False, g3 → False, w1 → False, w2 → False},
      OpAmpNoiseProduct[opamp, opampnoise, 7] /. {g12 → False,
         g6 → False, g3 → False, w1 → False, w2 → False}} //. ADA4898[f] /. fourkt],
  {f, 1, 10*^3}, PlotRange → {1, 50}, PlotLabel → "Input Referred Noise",
  FrameLabel → {"Hz", "nV/ √Hz "},
  Evaluate[plotoptn[6]],
  PlotLegends → mylegend[{"21dB Gain/2 whitening stages", "2 whitening stages",
     "1 whitening stage", "21dB Gain", "12dB Gain", "No gain/no whitening"}]]
```

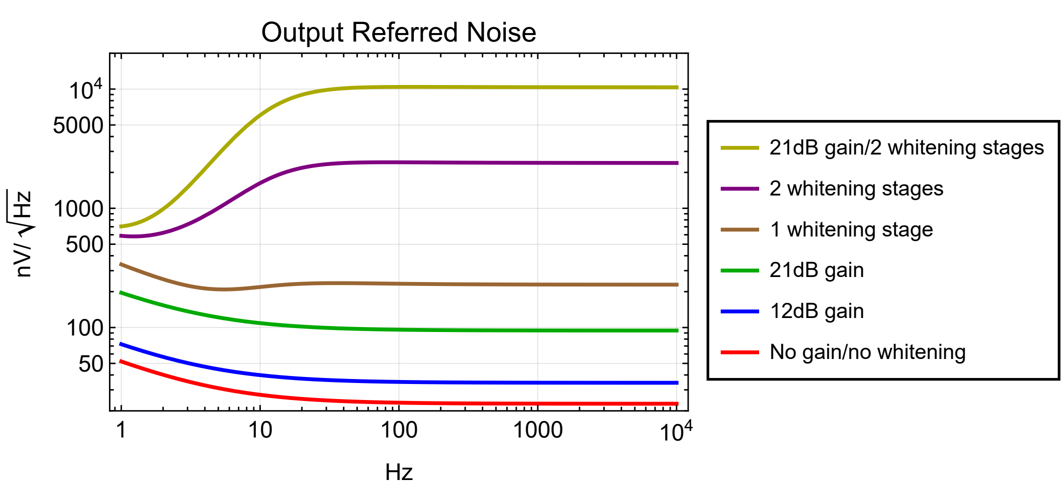
Out[ ]=



Output Referred Noise

In[ ]:=
```
LogLogPlot[Evaluate[1*^9 {
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /.
       {g12 → True, g6 → True, g3 → True, w1 → True, w2 → True},
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /.
       {g12 → False, g6 → False, g3 → False, w1 → True, w2 → True},
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /.
       {g12 → False, g6 → False, g3 → False, w1 → True, w2 → False},
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /.
       {g12 → True, g6 → True, g3 → True, w1 → False, w2 → False},
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /.
       {g12 → True, g6 → False, g3 → False, w1 → False, w2 → False},
      OpAmpNoiseProductOut[opamp, opampnoise, 7] /. {g12 → False,
        g6 → False, g3 → False, w1 → False, w2 → False}} //. ADA4898[f] /. fourkt],
 {f, 1, 10*^3}, PlotRange → {20, 20000}, PlotRange → {1, 50},
 PlotLabel → "Output Referred Noise",
 FrameLabel → {"Hz", "nV/ √Hz "},
 Evaluate[plotoptn[6]],
 PlotLegends → mylegend[{"21dB gain/2 whitening stages", "2 whitening stages",
    "1 whitening stage", "21dB gain", "12dB gain", "No gain/no whitening"}]]
```

Out[ ]=



ADAQ23876

r1 || r2: internal feedback resistor; r3: input resistor; ADCFS: full scale ADC range; FS: Input full scale; SNR: ADC SNR

r4: extra input series resistor needed

In[◦]:=
```
prmADAQ23876 = {r1 → 1000, r2 → 1375, r3 → 1571,
   ADCFS → 4.096, FS → 20, SNR → 10^(90.7/20), BW → 2^22, rfb → par[r1, r2]};
```

In[◦]:=
```
Solve[rfb/(r3 + r4) == ADCFS/FS, r4][[1]]
r4 /. % //. prmADAQ23876
```

Out[◦]=

$$\left\{r4 \to \frac{-ADCFS\ r3 + FS\ rfb}{ADCFS}\right\}$$

Out[◦]=

1255.89

In[◦]:=
```
adcnoise = 1*^9 FS/(Sqrt[2] SNR) 1/Sqrt[BW] /. prmADAQ23876 (* nV/Sqrt[Hz] *)
```

Out[◦]=

201.458