

---

# Deliver 01

This is the report title

---

Llorenç Fanals Batllori

llorenc.fanals@estudiantat.upc.edu



31/8/2021

ETSETB, UPC

Advanced Analog Circuit Techniques (AACT, 230642)

# Table of contents

<b>1</b>	<b>Exercises</b>	<b>2</b>
1.1	Exercise 1 . . . . .	2
1.2	Exercise 2 . . . . .	5
1.3	Schemes and plots . . . . .	8
1.4	Exercise 3 . . . . .	10
<b>2</b>	<b>Lab: Design and analysis of LNAs</b>	<b>14</b>
2.1	Introduction . . . . .	14
2.2	Creation of your Library and LNA cell . . . . .	14
2.3	First simulation of the circuit . . . . .	14
2.4	Evaluating Gains and Input Matching . . . . .	16
2.5	Evaluating Isolation and Stability . . . . .	23
2.6	Evaluating Noise performance . . . . .	24
2.7	Evaluating Linearity . . . . .	29
2.8	Performance Summary . . . . .	32
<b>3</b>	<b>Code</b>	<b>35</b>
3.1	VHDL . . . . .	35
3.2	C . . . . .	37

# 1. Exercises

## 1.1. Exercise 1

### Gm-C filters

We promised Gm-C filters to be an advantageous solution to overcome main limitations of RC-OpAmps filters, i.e. operation at high frequencies and operation with low power consumption. It's the moment now to make some numbers and check whether this is really true.

Assume a transconductance  $G_m = 1 \text{ mA}$ . Calculate the expected corner frequencies achievable in a 1<sup>st</sup>-order single-ended filter with capacitor values of 20 pF, 5 pF, 2 pF, 500 fF and 200 fF. Which is the reasonable range of frequencies in a microelectronic CMOS implementation? Would this frequency range be achievable in a RC-OpAmp implementation?

The section 1.1 contains the statement and solution of the exercise.

Considering the single-ended schematic shown at the slide number 52, one can prove that

$$\frac{v_{out}}{v_{in}} = \frac{-G_m}{sC_{int}} . \quad (1)$$

Thus, the pole frequency appears at

$$f_p = \frac{1}{2\pi} \frac{G_m}{C_{int}} . \quad (2)$$

From this formula, the different frequencies can be calculated, taking  $G_m = 1 \text{ mA/V}$ .

$C_{int}$	$f_p$
20 pF	7.96 MHz
5 pF	31.83 MHz
2 pF	79.58 MHz
500 fF	318.31 MHz
200 fF	795.77 MHz

Table 1. Pole frequency for the given  $C_{int}$  values

The OpAmps we have analyzed in the subject require a low-frequency pole when working closed-loop to get a decent phase margin. Also, they usually require buffers. These two factors make them only suitable for low-medium frequencies, so they can have a corner frequency (bandwidth) of a few MHz at best. Slide 56 suggests me that they are only suitable up to 10 MHz.

Thus, I guess that only the 7.96 MHz would be achievable. The larger frequencies would not be

possible, as an RC-OpAmp would not be capable of achieving this large bandwidths.

**Assume a 1<sup>st</sup> order Gm-C single-ended filter with an integrator capacitor of 300 fF. Calculate the expected corner frequencies with  $G_m$  values of 1 mA/V, 100  $\mu$ A/V, 10  $\mu$ A/V, 1  $\mu$ A/V and 100 nA/V. Assuming  $G_m = g_m$ , calculate the required DC current consumption to achieve these  $G_m$ 's, assuming an overdrive  $V_{OD} = 200$  mV. Which is the range of frequencies for which you get ultra-low currents  $< 1$   $\mu$ A? How can you further reduce the current to obtain a  $G_m = 100$  nA/V?**

Now, we are given different  $G_m$  values, for which the different corner frequencies can be calculated.

$G_m$	$f_p$	$I_{DQ}$
1 mA/V	530.51 MHz	100 $\mu$ A
100 $\mu$ A/V	53.05 MHz	10 $\mu$ A
10 $\mu$ A/V	5.30 MHz	1 $\mu$ A
1 $\mu$ A/V	530.51 kHz	100 nA
100 nA/V	53.05 kHz	10 nA

Table 2. Pole frequency and current consumption for the given  $C_{int}$  values

Thus, from the table, the range of frequencies for which we get ultra-low currents is

$$\boxed{0 \leq f \leq 5.3051 \text{ MHz}} \quad (3)$$

One would think that as  $g_m = \frac{2I_{DQ}}{V_{OD}}$ , there's a direct relation between both  $G_m$  and the current, and that if we desire to reduce the current consumption, then  $G_m$  must also be decreased. The thing is that this can be true while working in strong inversion.

Instead, in the weak inversion zone ( $V_{OD} < 0$  V), the  $g_m$  transconductance is at its maximum for a given drain current. In weak inversion,

$$g_m = \frac{I_D}{\eta V_T} . \quad (4)$$

For instance, in the slides "MOS\_reminder",  $\eta = 1.5$ . Taking a typical value for the thermal voltage,  $V_T = 0.026$  V, the factor that multiplies  $I_D$  is  $\approx 25$ .

If the same transistor was in strong inversion,  $V_{OD} > 200$  mV. Taking this limit value, we would get  $g_m = 10I_D$ . The  $I_D$  factor would be even lower for larger  $V_{OD}$  values.

Thus, the  $g_m$  transconductance can easily be higher in weak inversion for the same  $I_D$  current.

This is interesting for low-power applications.

## 1.2. Exercise 2

### Gm-C integrator

Consider this Gm-C integrator. Assume  $C_{INT} = 600 \text{ fF}$ ,  $k' = 200 \mu\text{A/V}$ ,  $W/L$  of M1-M2 = 10 and  $W/L$  of M5 = 20. Assume  $V_T = 0.5 \text{ V}$  and all transistors biased such that  $V_{OD} = 0.25 \text{ V}$ . An additional CMFB circuit (not drawn) sets the output common-mode to  $2.5 \text{ V}$ .

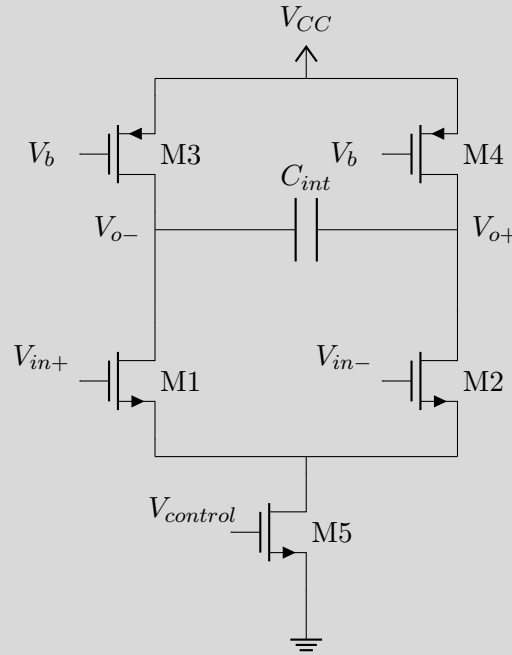


Figure 1. Gm-C integrator

Which is the corner frequency that can be obtained from this integrator?  
Which is the range of input common-mode voltages?

First, one can see that both M3 and M4 are set with a constant voltage, independent of  $V_{in+}$  and  $V_{in-}$ , that is, independent of  $v_{in}$ . Thus, if  $V_{in+}$  raises by  $v_d/2$ , then  $V_{in-}$  diminishes by the same value. The current increase at M1 is provided through  $C_{int}$ , and its value is equal to the current decrease in M2. The current increase at M1 is

$$i_{d1} = \frac{v_d}{2} g_m \quad (5)$$

The voltage difference between the output terminals is

$$\begin{aligned} v_{out} &= i_{d1} Z_{C_{int}} \\ v_{out} &= \frac{v_d}{2} g_{m_{1,2}} \frac{1}{s C_{int}} \end{aligned} \quad (6)$$

Then,

$$\frac{v_{out}}{v_d} = \frac{1}{2} \frac{g_{m_{1,2}}}{s C_{int}} \quad (7)$$

Thanks to the given data, the  $I_{M1}$  current can be calculated.

$$I_{M1} = \frac{k'}{2} \frac{W}{L} V_{OD}^2 . \quad (8)$$

$$I_{M1} = 62.5 \mu\text{A}$$

From this,

$$g_m = 500 \mu\text{A/V} . \quad (9)$$

The frequency for which the gain is 1,

$$f_{corner} = \frac{1}{2\pi} \frac{g_{m1,2}}{2C_{int}} . \quad (10)$$

Which equals

$$\boxed{f_{corner} = 66.31 \text{ MHz}} \quad (11)$$

Given that  $V_{OD} = 0.25 \text{ V}$  for all transistors, the minimum  $V_{in \text{ CM}}$  is

$$V_{in \text{ CM min}} = V_{DS_{M5 \text{ min}}} + V_{OD} + V_T . \quad (12)$$

Where  $V_{DS_{M5 \text{ min}}} = V_{OD} = 0.25 \text{ V}$ . Thus,

$$\boxed{V_{in \text{ CM min}} = 1 \text{ V}} \quad (13)$$

The maximum is set by the CM voltage at the output, which happens to be  $2.5 \text{ V}$ . Then,

$$V_{in \text{ CM max}} = (V_{OUT} - V_{OD}) + V_{OD} + V_T . \quad (14)$$

So,

$$\boxed{V_{in \text{ CM max}} = 3 \text{ V}} \quad (15)$$

With all this,

$$\boxed{1 \leq V_{in \text{ CM}} \leq 3 \text{ V}} \quad (16)$$

**Imagine now we want to benefit from the tunable feature with  $V_{control}$  in order to produce a filter with a corner frequency adjustable over one decade (i.e. from 1 to 10 times the frequency you obtained in the former section). Which is the range of  $V_{control}$  values needed to achieve this frequency range? Which is now the range of input common-mode voltages, for the highest-frequency situation?**

From the lecture notes,

$$g_m = k' \sqrt{(W/L)_1 (W/L)_5} \frac{1}{2} (V_{control} - V_T) . \quad (17)$$

Where, according to the previous section,  $V_{control} - V_T = V_{OD} = 0.25 \text{ V}$ .

Remember from (1.2) that there's a linear relation between  $g_m$  and  $f_{corner}$ . Thus, if  $f_{corner}$  can be increased by a factor up to 10, so does  $g_m$ . Thus, for this limit case,  $V_{control} - V_T$  must also be 10 times greater, that is,  $V_{control} = 3\text{ V}$ .

Notice that to achieve the corner frequency calculated,  $V_{OD} = 0.25\text{ V}$ ,  $V_{control} = 0.75\text{ V}$ . Then, the range to have a corner frequency from 1 to 10 times the previously calculated one,

$$\boxed{0.75\text{ V} \leq V_{control} \leq 3\text{ V}} \quad (18)$$

For the maximum  $V_{control}$  voltage of the previous equation, one can see that  $V_{OD} = 2.5\text{ V}$ . Thus, the minimum voltage at the source of M1 must be also  $2.5\text{ V}$ . This would give place to  $V_{IN\ CM} = 3.25\text{ V}$ , which is not possible, because the CMFB is setting the output nodes at  $2.5\text{ V}$ .

Because of this, the maximum voltage at the drain of M5 is  $2.25\text{ V}$ , for which only up to 7 times the calculated corner frequency is possible.



### 1.3. Schemes and plots

First, let's derive the output of both systems so as to be able to easily solve the 4 questions that appear on the next section.

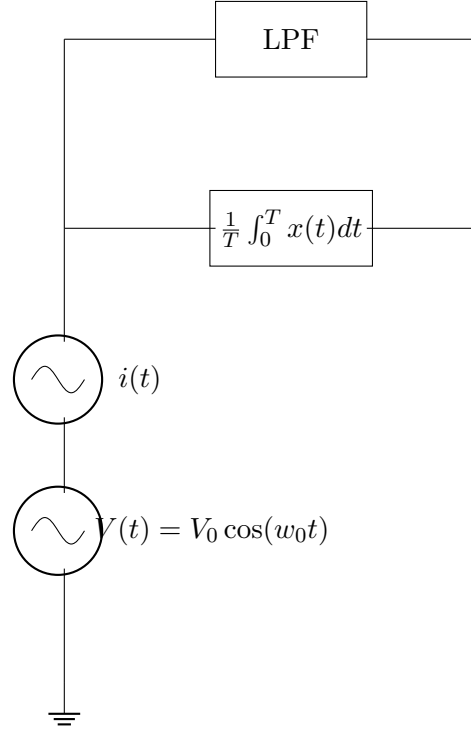


Figure 2. Schematic of the exercise

Where the integrator performs

$$y(n) = \frac{1}{T} \int_0^T x(t) dt . \quad (19)$$

Where  $T$  is the integration time. When the input to this block is  $V(t) = V_0 \cos(w_0 t + \phi) = V_0 \sin(w_0 t + \phi_2)$ , where  $\phi_2 = \frac{\pi}{2} + \phi$ , we have

$$y(n) = \frac{1}{T} \int_0^T V_0 \sin(w_0 t + \phi_2) = V_0 \frac{1}{w_0 T} \frac{1}{T} [\cos(w_0 T + \phi_2) - \cos(\phi_2)] . \quad (20)$$

The idea here is to manipulate the expression so that we can express the output in terms of the input and a  $\sin()$  or  $\cos()$  function. In the slides, the previous equation is rewritten as

$$y(n) = \frac{2}{w_0 T} V_0 \sin\left(\frac{w_0 T}{2} + \phi_2\right) \sin\left(\frac{w_0 T}{2}\right) . \quad (21)$$

To make sure this is correct, trigonometric functions can be expanded using Euler's identity. We want to prove

$$\cos(w_0 T + \phi_2) - \cos(\phi_2) = 2 \sin\left(\frac{w_0 T}{2} + \phi_2\right) \sin\left(\frac{w_0 T}{2}\right) . \quad (22)$$

This previous equation becomes

$$\begin{aligned} \frac{e^{j(w_0T+\phi_2)} + e^{-j(w_0T+\phi_2)}}{2} - \frac{e^{j\phi_2} + e^{-j\phi_2}}{2} &= 2 \frac{e^{j\frac{w_0T}{2}} - e^{-j\frac{w_0T}{2}}}{2} \frac{e^{j(\frac{w_0T}{2}+\phi_2)} - e^{-j(\frac{w_0T}{2}+\phi_2)}}{2} \\ e^{j(w_0T+\phi_2)} + e^{-j(w_0T+\phi_2)} - e^{j\phi_2} + e^{-j\phi_2} &= \left( e^{j\frac{w_0T}{2}} - e^{-j\frac{w_0T}{2}} \right) \left( e^{j(\frac{w_0T}{2}+\phi_2)} - e^{-j(\frac{w_0T}{2}+\phi_2)} \right) . \\ e^{j(w_0T+\phi_2)} + e^{-j(w_0T+\phi_2)} - e^{j\phi_2} + e^{-j\phi_2} &= e^{j(w_0T+\phi_2)} + e^{-j(w_0T+\phi_2)} - e^{j\phi_2} + e^{-j\phi_2} \end{aligned} \quad (23)$$

So, yes, (24) is correct. Now,  $\sin()$  is transformed back to  $\cos()$ .

$$y(n) = \frac{2}{w_0T} V_0 \cos\left(\frac{w_0T}{2} + \phi\right) \sin\left(\frac{w_0T}{2}\right) . \quad (24)$$

This is the output out of the integrator. It can also be written as

$$y(n) = \frac{1}{\pi fT} V_0 \cos(\pi fT + \phi) \sin(\pi fT) = V_0 \cos(\pi fT + \phi) \frac{\sin(\pi fT)}{\pi fT} . \quad (25)$$

Where  $f$  is the frequency of  $w_0$ , i.e., the frequency of the signal. The  $\frac{\sin(\pi fT)}{\pi fT}$  is the sinc function, so it can output values of up to 1, and 0 when  $fT$  is an integer number.

One can wonder what's the output of the LPF. In this case, we have to keep in mind this system shows the same bandwidth as for the discrete integrator. Then, the output of this system must be tangent to the output of the discrete integrator only in the points in which the discrete integrator output is maximum. There's a plot about SMRR on the slides that can be of help in visualizing this.

I plot the derived expression and the expression of the other system, without multiplying them by the input signal. The module is taken.

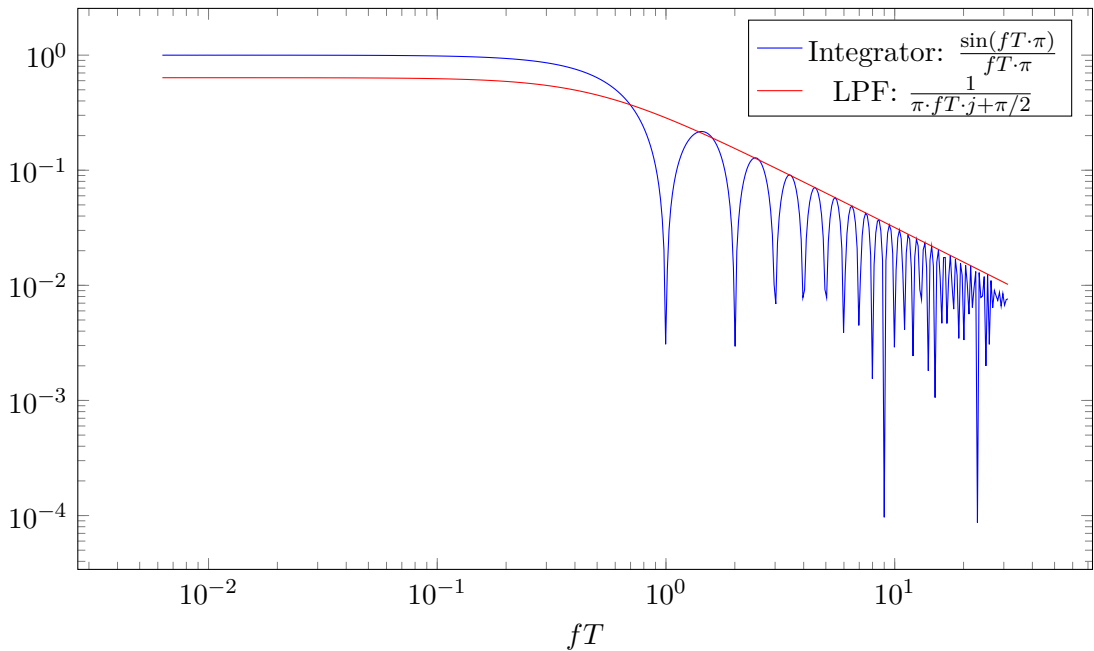


Figure 3. Output of the 2 integrators

The LPF equation has been derived the following way. Initially, I wrote the following generic low pass filter equation:

$$LPF = \frac{k}{b \cdot fTj + a} . \quad (26)$$

Where  $k$ ,  $b$ , and  $a$  are constants to determine, and  $j = \sqrt{-1}$ . First, I wanted to be compliant with the specified bandwidth  $BW = \frac{1}{2T}$ . Second, I wanted the same slope for high frequencies. Thus,

$$\begin{aligned} \frac{k}{b} &= \frac{1}{\pi} \\ f_{-3dB} &= \frac{1}{2T} . \\ f_{-3dB} &= \frac{a}{bT} \end{aligned} \quad (27)$$

With this, and setting  $k = 1$ , I get

$$\begin{aligned} b &= \pi \\ a &= \frac{\pi}{2} . \end{aligned} \quad (28)$$

Leading to

$$LPF = \frac{1}{\pi fTj + \frac{\pi}{2}} . \quad (29)$$

Notice that this is the exact equation that has been plotted above, and that for  $-3$  dB both plots intersect, meaning they have the same bandwidth. The bandwidth  $BW = \frac{1}{2T}$  was a specification given in the statement. As it has been proven, it's the correct value to have the two bandwidths to be equal.

So, what's the output of the LPF for a sinusoidal input  $V(t) = V_0 \cos(w_0t + \phi) = V_0 \cos(2\pi ft + \phi)$ ? Well, taking into account the gain and the phase of the system,

$$y = V_0 \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi fT)^2}} \cos(2\pi ft + \phi - \arctan(4\pi fT)) . \quad (30)$$

Now, we know how will be the output of both systems when a cosine with a certain frequency and amplitude is applied. We are in a position to solve the next questions. If  $w \geq 0$ , we will just have to plug this into the outputs of the two systems. If there's a signal but also an interference, we can apply the superposition theorem.

#### 1.4. Exercise 3

**a) For  $i(t) = 0$  and  $w_0 = 0$  demonstrate that the amount information out of both systems is the same.**

In this case, there's no interference. It's enough to evaluate the output of both systems for the

same applied input signal,  $V(t) = V_0 \cos(w_0 t)$ . We have already analyzed the output of both systems for a signal like this.

$$\begin{aligned} y_1 &= V_0 \cos(\pi f T + \phi) \frac{\sin(\pi f T)}{\pi f T} \\ y_2 &= V_0 \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f T)^2}} \cos(2\pi f t + \phi - \arctan(4\pi f T)) \end{aligned} \quad (31)$$

In this case,  $\phi = 0$ . Now, we can perform the calculations for  $f \rightarrow 0$ . The identity  $\frac{\sin(x)}{x}|_{x \rightarrow 0} \approx \frac{x}{x} \approx 1$  can be of help. Then,

$$\begin{aligned} y_1|_{f \rightarrow 0} &= V_0 \cos(\pi f T + \phi) \frac{\sin(\pi f T)}{\pi f T} \approx V_0 \\ y_2|_{f \rightarrow 0} &= V_0 \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f T)^2}} \cos(2\pi f t + \phi - \arctan(4\pi f T)) = 4V_0 \end{aligned} \quad (32)$$

Recall  $\phi = 0$ . The output of the integrator could already be deduced from the above plot without problems. While it's harder to appreciate on the plot the output of the LPF for this case, it's easy to verify that it must be equal to  $4V_0$ . The LPF equation can be analyzed for  $f \rightarrow 0$  and the 4 factor arises.

What's important to realise is that at both outputs the interference, or the noise, will be  $N = 0$ . The information formula is

$$I = B \log_2 \left( 1 + \frac{S}{N} \right) \approx 3.32 B \log_{10} \left( 1 + \frac{S}{N} \right) \quad (33)$$

As both noises are 0,

$$\boxed{I_1 = I_2 = \infty} \quad (34)$$

Keep in mind this is a theoretical limit. In practice, this will never be achieved.

**b) For  $i(t) = 0$  and  $w_0 > 0$  demonstrate that the amount information out of both systems is the same. Are there any restrictions on the relationship  $T$  and  $w_0$ ?**

Going back to the equations, notice that for  $f > 0$  the output formulas we obtained previously should be analyzed carefully:

$$\begin{aligned} y_1 &= V_0 \cos(\pi f T + \phi) \frac{\sin(\pi f T)}{\pi f T} \\ y_2 &= V_0 \cos(2\pi f t + \phi - \arctan(4\pi f T)) \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f T)^2}} \end{aligned} \quad (35)$$

As long as  $fT \notin \mathbb{Z}$ , i.e., the  $fT$  factor is not an integer, then  $\sin(\pi f T) \neq 0$ , so there's a signal out of the first system.

For the second system, the LPF, for certain instants of time  $t$  there's the risk to get a 0 output. However, what's of interest here is to notice the output signal will be attenuated a certain amount, as the plot suggests. Of course, for the integrator, if we had integrated from  $t_1$  to  $t_2$  instead of from 0 to  $T$ , we would also depend on these values of time to assure whether the output is zero or not. The key idea is to not be bothered about the value the input signal takes at the input, but to get an idea about the relation between input and output (gain). For this, the plot can be of great help.

Given that there's no noise, it's direct to see that

$$I_1 = I_2 = \infty \quad (36)$$

Recall this is only true if

$$\frac{w_0}{2\pi}T \notin \mathbb{Z} \quad (37)$$

This condition guarantees that we don't integrate over a whole number of periods, as for this case the integral would be 0. This is a very important condition that could be derived for the plot.

On the other hand, if it happens that the  $\cos()$  of  $y_2$  is 0 for a certain instant of time, we shouldn't bother much, as for other instants of time we will get a signal. For the LPF the signal suffers attenuation and phase shift, but in any case its gain goes to 0, i.e., attenuation goes to  $\infty$ . This is a key concept that can be useful for d) section.

**c) For  $i(t) = n(t)$ ,  $w_0 > 0$  and the above restrictions, demonstrate that the amount of information out of both systems is the same.**

So now, thanks to "the above restrictions", we can assume the frequency and the period of integration product don't result in an integer number.

Notice that now we have white Gaussian noise, which means that in the frequency spectrum we have a constant PSD from  $-\infty$  to  $\infty$ , and in the time domain it follows a Gaussian distribution of 0 mean. It is well known that the integration of this noise must lead to an average value of 0<sup>1</sup>. Of course, there will be some uncertainty in this  $N = 0$  (noise equals 0) average value, but here we are looking at average values. Then, provided  $\frac{w_0}{2\pi}T \notin \mathbb{Z}$ ,

$$I_1 = I_2 = \infty \quad (38)$$

**d) What happens if  $i(t) = V_i \cdot \sin(w_i t)$ ?**

---

<sup>1</sup>MIT notes on noise.

Now, the interference is no longer white Gaussian noise. We can apply the superposition theorem and the previous expressions to affirm that

$$\begin{aligned}
 y_1 &= V_0 \cos(\pi f T + \phi) \frac{\sin(\pi f T)}{\pi f T} + V_i \cos(\pi f_i T + \phi) \frac{\sin(\pi f_i T)}{\pi f_i T} \\
 y_2 &= V_0 \cos(2\pi f t + \phi - \arctan(4\pi f T)) \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f T)^2}} \\
 &\quad + V_i \cos(2\pi f_i t + \phi - \arctan(4\pi f_i T)) \frac{1}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f_i T)^2}}.
 \end{aligned} \tag{39}$$

Where  $f_i$  is the frequency of the interference signal and  $V_i$  its amplitude, and  $f$  is the frequency of the signal and  $V_0$  its amplitude. Each one of the informations is depicted here:

$$\begin{aligned}
 I_1 &= B \log_2 \left( 1 + \frac{V_0 \cos(\pi f T + \phi) \frac{\sin(\pi f T)}{\pi f T}}{V_i \cos(\pi f_i T + \phi) \frac{\sin(\pi f_i T)}{\pi f_i T}} \right) \\
 I_2 &= B \log_2 \left( 1 + \frac{\frac{V_0 \cos(2\pi f t + \phi - \arctan(4\pi f T))}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f T)^2}}}{\frac{V_i \cos(2\pi f_i t + \phi - \arctan(4\pi f_i T))}{\sqrt{\left(\frac{1}{4}\right)^2 + (\pi f_i T)^2}}} \right).
 \end{aligned} \tag{40}$$

For  $y_1$ , i.e., the output out of the discrete integrator, we could set  $f_i T \in \mathbb{Z}$  so that the interference is totally eliminated. Then, we could still get  $I_1 = \infty$ , theoretically. In reality, noise will never be totally eliminated, though, but we can potentially get a very nice SNR (signal-to-noise ratio).

For  $y_2$ , which is the output of the LPF, we can't attenuate the noise. In this case, probably the signal level will be higher than for  $y_1$ , but there's no way to effectively attenuate only the noise.

All in all, if the  $T$  value is adjusted so as to eliminate the noise interference, and  $f_i$  is known, one can expect a much higher theoretical information bound for  $y_1$  than for  $y_2$ ,  $I_1 > I_2$ . The previous plot shows how this technique can be very effective in improving the signal-to-noise ratio, if  $T$  is adjusted correctly,  $f_i$  is known, and there's some separation between the interference and the signal frequencies.

## 2. Lab: Design and analysis of LNAs

### 2.1. Introduction

### 2.2. Creation of your Library and LNA cell

### 2.3. First simulation of the circuit

Capture the windows containing your design selections (Schematic, Variables section of the ADE window). Include them in your report.

Although my pre-lab calculations were pretty close to the ones provided by the professor, I ended up using his numbers, as we were told they would let us obtain very good results without tuning much the circuit.

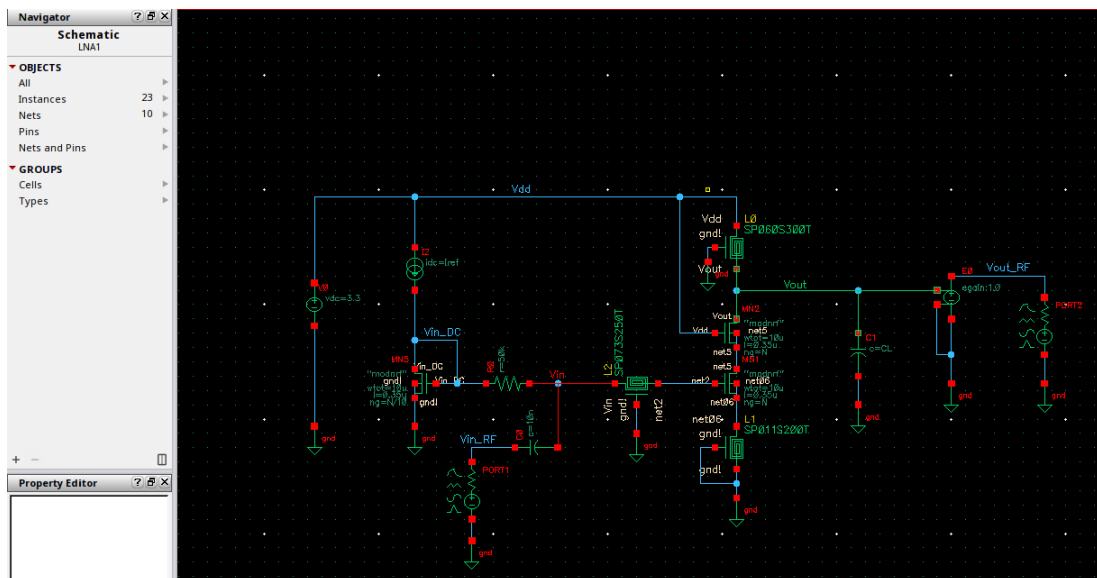


Figure 4. Schematic

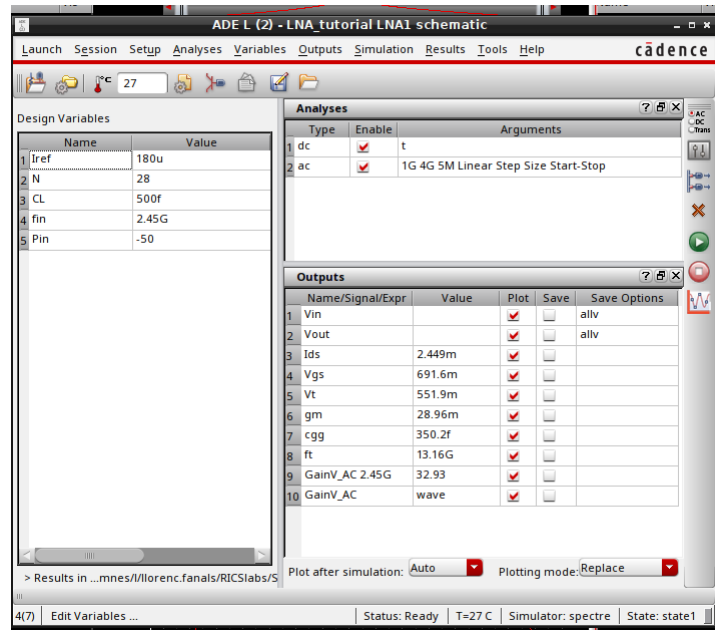


Figure 5. ADE window

Create a small table comparing the OP parameter values against the values expected according to your prelab calculations. Include them in your report.

As commented, I use the numbers provided by the professor, and for  $V_{TH}$  I take the value that appears on the technology data table.

Characteristic	Theoretical value	Simulation value
$I_{DC}$	2.5 mA	2.449 mA
$V_{GS}$	0.6822 V	0.6961 V
$V_{TH}$	0.5 V	0.5519 V
$g_m$	24.7 mA/V	28.96 mA/V
$C_{GG}$	350 fF	350.2 fF
$f_T$	11.2 GHz	13.16 GHz

Table 3. Parameter comparison

The results are similar in most cases. The transconductance  $g_m$  is higher for the simulation, thus leading to a higher  $f_T$  value, too. This is something positive, we can get a higher gain for the same power consumption.

Capture the results of the AC analysis, including markers measuring the resonance frequency and voltage gain.



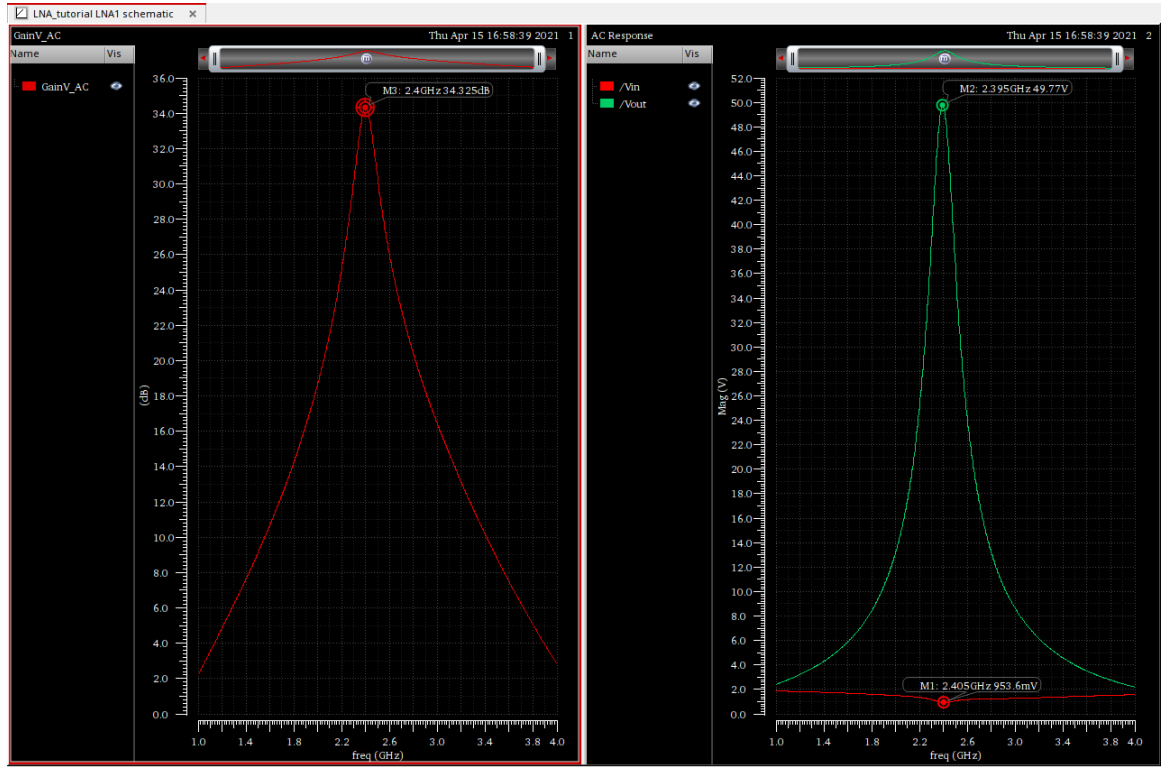


Figure 6. Gain and voltage plots

$$|A_v| = 34.325 \text{ dB} . \quad (41)$$

Can you extract some information on the input matching, from the plots obtained?

I extract that input matching must be very good because the obtained gain is higher than  $A_v = 30 \text{ dB}$ , which was the gain we pursued in the pre-lab. Part of this comes from the fact that the  $g_m$  obtained is greater than the theoretical one, but in spite of this we wouldn't obtain this large gain if it was not for very good input matching, which is evaluated in the following section.

## 2.4. Evaluating Gains and Input Matching

Capture the plot of the S11 parameter, including markers to measure the resonance frequency of S11 and its value at  $f_0 = 2.45 \text{ GHz}$ .

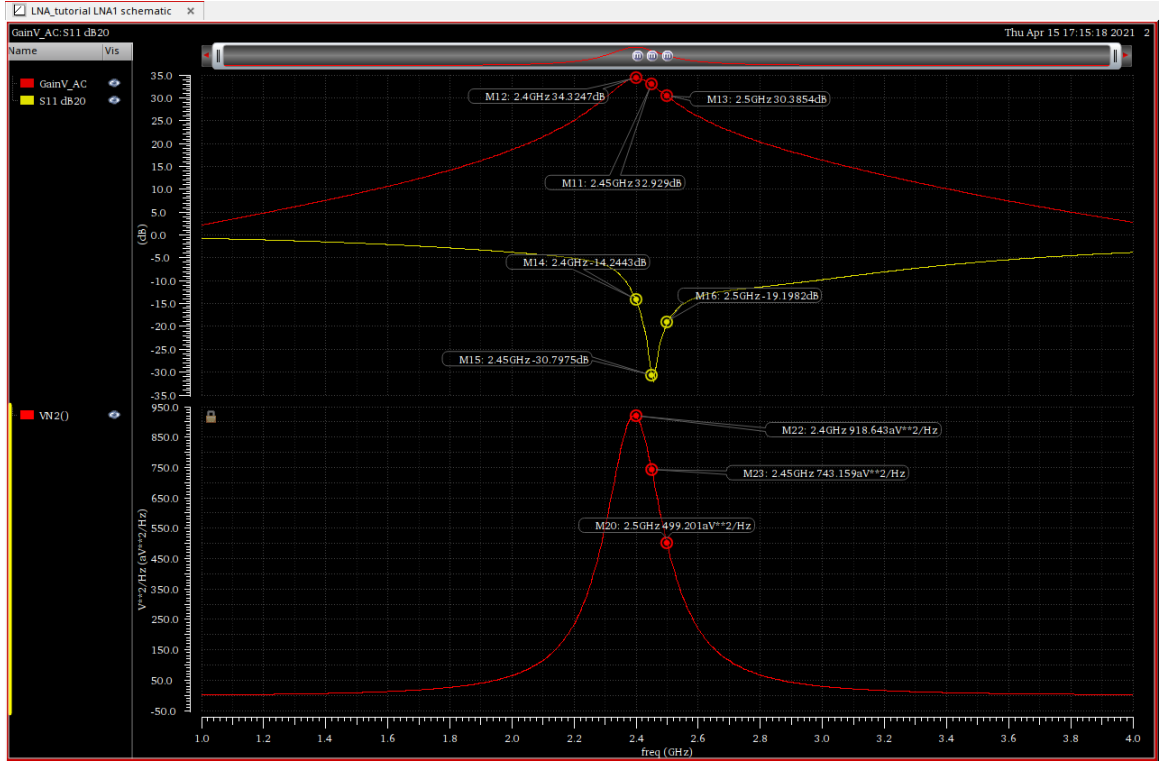


Figure 7. Gain and S11 plots

While  $f = 2.45$  is not the frequency for which the gain is at its maximum, it is the frequency for which the input matching is best.

$$S_{11}|_{2.45 \text{ GHz}} = -30.7975 \text{ dB} . \quad (42)$$

This fulfils the  $S_{11}$  specification without a problem, which consisted on  $S_{11} < -10$  dB.

Around  $f_0$ , the central frequency, the S11 parameter shows very small values. This means that the ratio between the reflected wave and the incident wave is very low (if perfectly isolated, as is the case thanks to the buffer, S11 is just this relation). This is something desired because it means that almost no power will be lost in the input stage, so it can be concluded impedance matching is very good.

Recall,

$$\begin{pmatrix} V_1^- \\ V_2^+ \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} V_1^+ \\ V_2^- \end{pmatrix} . \quad (43)$$

**Capture the plots of the impedance, including markers at  $f_0 = 2.45$  GHz.**

Now, the input impedance is obtained, both the real and imaginary parts. Recall that, ideally, the imaginary impedance should be 0 at  $f_0$  and the real impedance should be 50  $\Omega$ . The closer

to these values, the better.

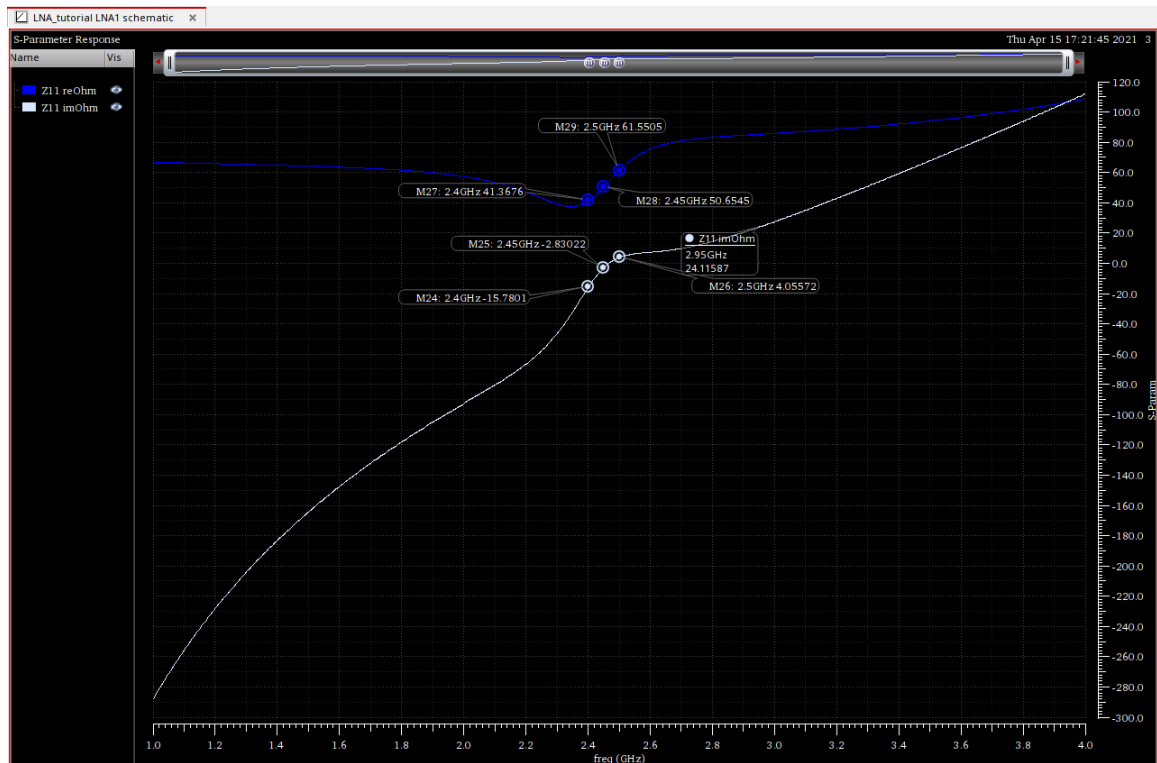


Figure 8. Z11 plots

Notice the real impedance at 2.45 GHz is very close to 50  $\Omega$ , and that the imaginary part is  $-2.83022j \Omega$ . So, the source impedance is very close to being matched by the amplifier input impedance. This explains the good  $S_{11}$  parameter.

**Compare the values obtained, to the targeted ones.**

Ideally, the amplifier input impedance should be equal to the voltage source output impedance: a real 50  $\Omega$ . Matching is not perfect but very close to it. By using the professor's numbers, I avoid having to tune the design. I already get a very low  $S_{11}$  parameter and a gain above 30 dB.

Also, I've obtained the Smith chart:

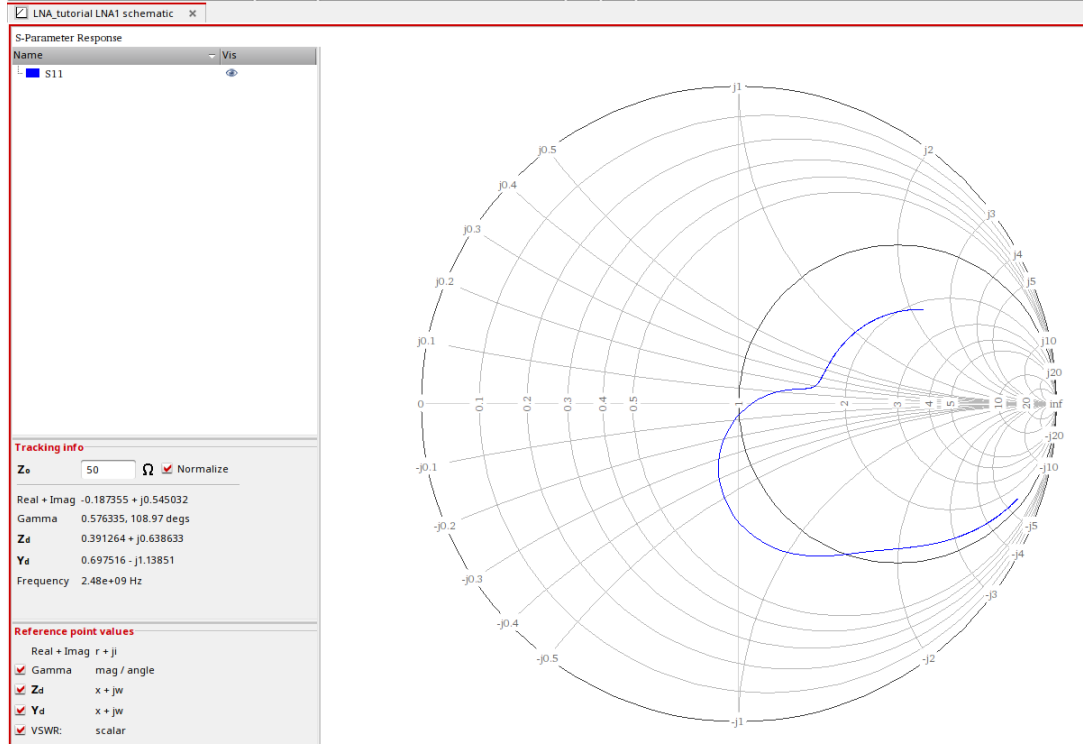


Figure 9. Smith chart

Notice the plot, in blue, crosses the unit circle around  $-0.05j$ . Considering it is normalized to  $50 \Omega$ , as commented on the laboratory document and as appears in the plot itself, this would lead to  $\approx -2.5j$ , which is very close to the actual obtained value,  $-2.83022j \Omega$ . As the cross is not in the center of the plot, matching is not perfect, but it's quite close to it.

**Capture the plots, including markers at  $f_0 = 2.45 \text{ GHz}$ .**

Now, we are interested in the gain from input to output. Notice from the previous equation that  $S_{21}$  relates the  $V_2^+$  to  $V_1^+$ .  $V_2^-$  in theory also affects, but in principle  $S_{22}$  is very low. The transducer power gain,  $G_T$ , and the operating power gain  $G_P$ , are

$$G_T = 10 \log \left( \frac{P_{OUT}}{P_A} \right) = 10 \log(|S_{21}|^2)$$

$$G_P = 10 \log \left( \frac{P_{OUT}}{P_{IN}} \right) = G_V + 10 \log \left( \frac{R_{in}}{R_L} \right) = 10 \log \left( \frac{|S_{21}|^2}{1 - |S_{11}|^2} \right) \quad (44)$$

If  $|S_{11}|$  is very low, which as seen previously is true, and can approximate  $S_{11} \approx 0$ , then  $G_T = G_P$ , i.e., the trasducer power gain is equal to the operating power gain. For frequencies far from the ressonant frequency, high differences can be expected between  $G_T$  and  $G_P$ , because  $S_{11}$  is not longer approximately 0, which means that input matching is no longer close to perfection. Next, the  $S_{21}$ ,  $G_T$  and  $G_P$  plots are shown.

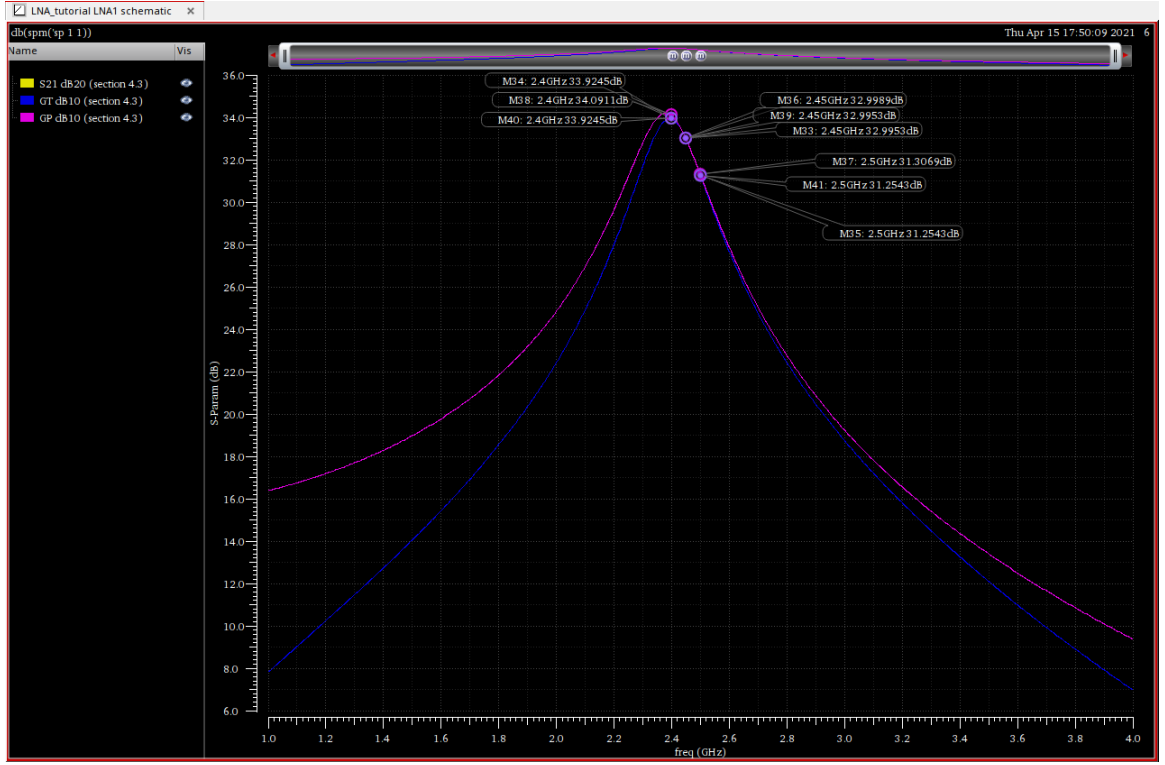


Figure 10. GT, GP, S21 plots

The  $S_{21}$  plot is overlapped with the  $G_T$  plot. It makes sense according to the definition I've just given. When there is not a good input matching (for frequencies far from the central one), the input power is lower than the one provided by the voltage source and thus  $G_P > G_T$ , because there's no perfect impedance matching, and thus  $P_{IN} < P_A$ . The  $G_P$  resembles a lot the gain plot I showed before. Anyways, the differences between the three plots are minimal in the expected frequencies.

**Are the results obtained consistent with the theoretical expectations?**

Yes, when there's good input matching (around the central frequency), the gain is over 30 dB, as desired. And when there's not good input matching, which is something that happens for frequencies far from the central one, the gain is much lower. Also, we know  $G_P$  is more "optimistic" than  $G_T$ ; the port component provides the specified power to the amplifier, at the cost that far from the central frequency  $P_A$  must be considerably higher than  $P_{IN}$ , i.e., not all the available power at the source arrives at the input of the amplifier.

Now, I've changed the PORT2 impedance to 500  $\Omega$ . Keep in mind the original schematic: the output port, PORT2, was connected to a 4-port component that copied the voltage at its two input ports to its two output ports, i.e., a simple buffer. So, the gain shown by the amplifier itself, defined as the output voltage divided by the input voltage, should not change in spite of increasing the output impedance.

Measure the four gain values at  $f_0 = 2.45$  GHz.

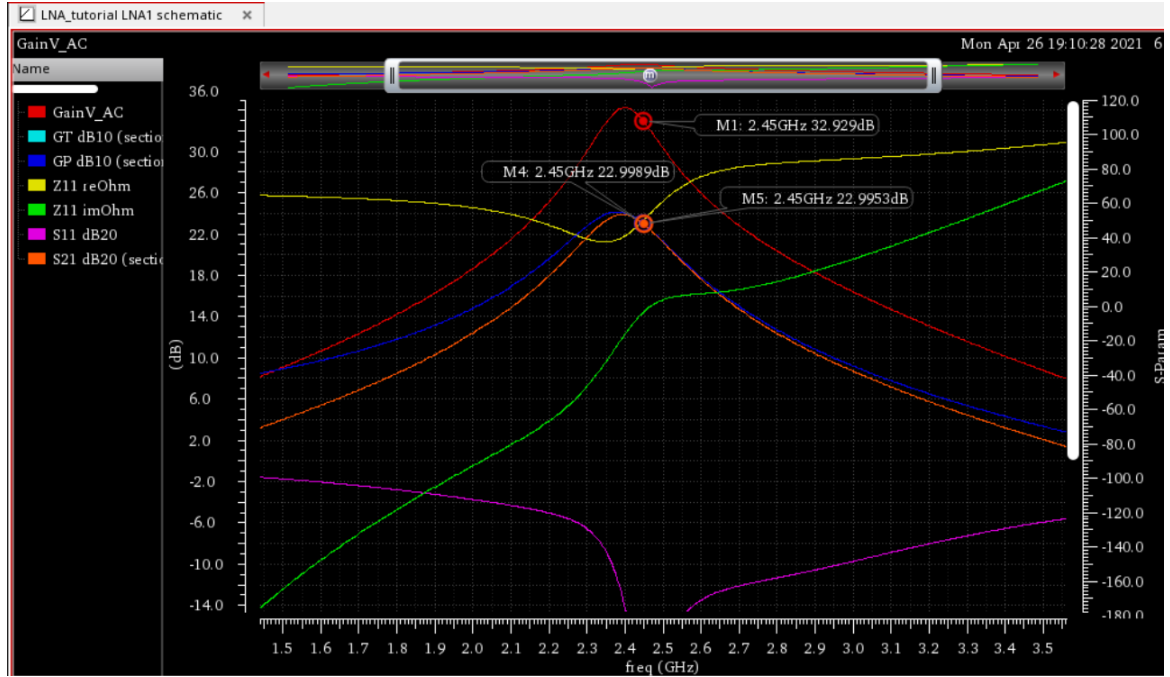


Figure 11. Plots

Again, the  $S_{21}$  plot is overlapped to the  $G_T$ .

Are the results obtained consistent with the theoretical expectations?

By increasing the output impedance by a 10 factor, the  $10 \log \left( \frac{R_{in}}{R_L} \right)$  term decreases by 10 dB. This happens to  $G_P$ , clearly from the equation, and also to  $|S_{21}|$  and thus  $G_T$ . This is the reason for the 10 dB shift.

Next, I've added a  $100 \Omega$  series resistance with  $L_G$ .

Capture the plots of the four gains, and the S11 parameter, including markers at  $f_0 = 2.45$  GHz.

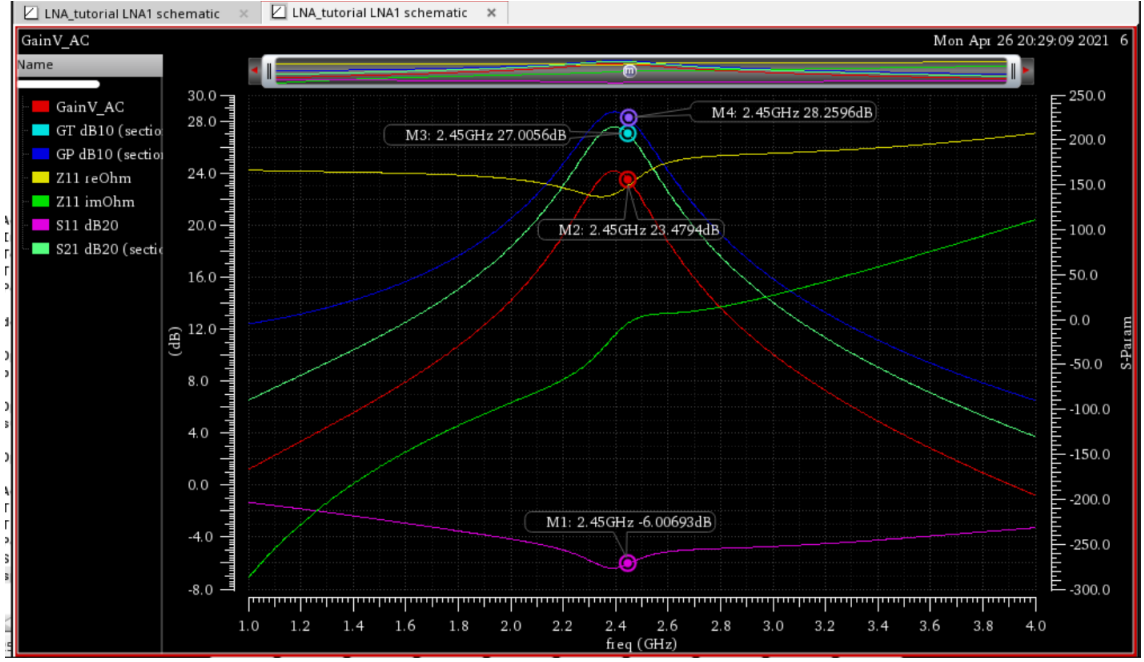


Figure 12. Plots

Again, the  $S_{21}$  plot is overlapped to the  $G_T$ .

**Are the results obtained consistent with the theoretical expectations?**

Now, there's no input matching at the central frequency. Recall we've verified our circuit was quite good at showing a real  $50\ \Omega$  input impedance at  $f_0$ . Now, by adding a resistance of  $100\ \Omega$  in series with the gate inductance, clearly  $S_{11}$  won't be as low as before, meaning there will be a partial reflection at the input and thus the output power will be lower. At the same time, this will mean the output gain will be lower.

Before, it was shown that we had  $S_{11} = -30.7975\ \text{dB}$ . Now, this is just  $-6\ \text{dB}$ . For the  $G_V$  gain, I notice the linear factor between the previous gain and the current one is very close to 3. It makes sense if one considers that only  $1/3$  of the voltage at the input node reaches the circuit we had before, i.e.,  $2/3$  of voltage fall at the  $100\ \Omega$  resistance.

$G_T$  and  $G_P$  also result affected by this. From previous expressions, it can be seen that  $G_T - 10\log(1 - |S_{11}|^2) = G_P$ . The plotted  $S_{11}^2 = -6\ \text{dB} = 0.2512$ , so  $10\log(1 - |S_{11}|^2) = 0.7488 = -1.256\ \text{dB}$ , which is almost exactly the difference  $G_P - G_T$ . The  $G_P$  gain has also been reduced  $-6\ \text{dB}$  from its original value, because of bad input matching.

## 2.5. Evaluating Isolation and Stability

Capture the plot evaluating the reverse isolation, including marker at  $f_0 = 2.45$  GHz.

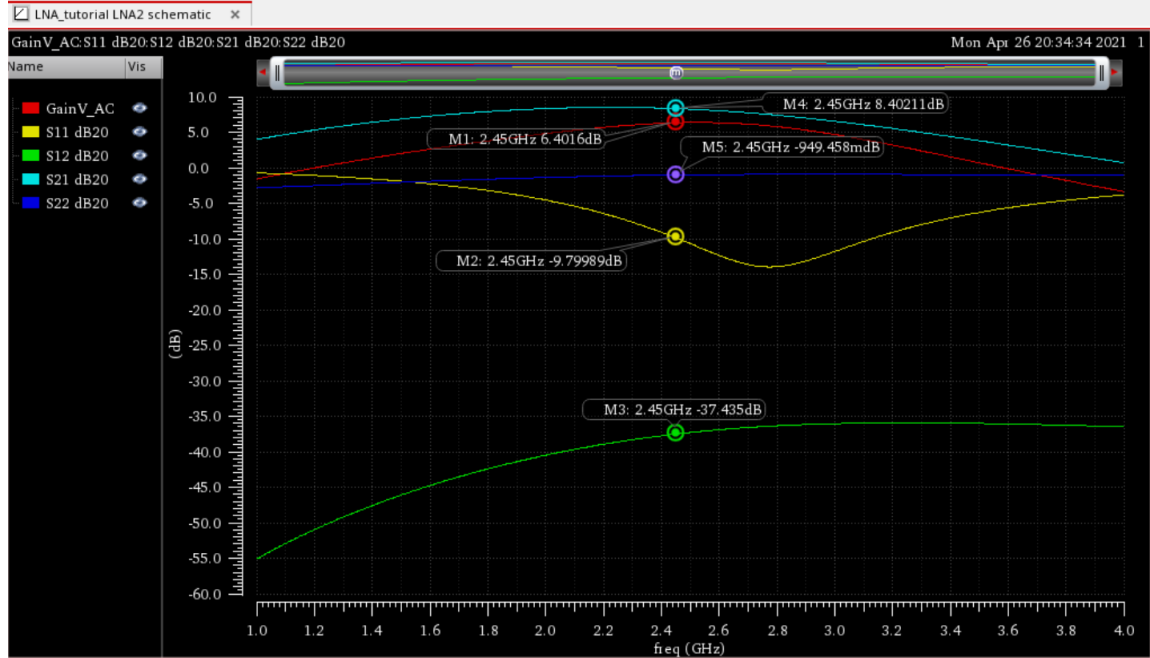


Figure 13. S parameters and gain plots

Does the reverse isolation have a good value?

Although it could always be lower, in a linear scale  $S_{12} \approx 0.014$ , so the contribution to  $V_1^-$  by the output is almost "only" 1% of the output voltage. However, keep in mind the output voltage is amplified by around 30 dB (in fact, from simulations it was around 33 dB). So, the input voltage is amplified by the amplifier, and then part of this amplified signal is feedback to the input. The feedback signal is  $\approx -4$  dB with respect to the input signal.

We already have the cascode transistor that helps quite a lot in isolating the output and the input (mainly the Miller capacitance that would appear with a single transistor does not longer appear with the cascode). So, I don't think there's much more to do to improve the  $S_{12}$  parameter. Keep in mind the comment made in the document lab, it suggests that this output loading would not happen in a real situation, so we should not bother much about it.

Also, check the consistency of the other S-parameters.

I think the  $S_{11}$  is not as low as before, but in the beginning we were required to get  $S_{11} < -10$



dBm, and in this case I almost get it. Impedance matching could be better, but it's decent. Of course,  $S_{21}$  shows the higher values.

Capture the plots of stability parameters, including markers at minimum/-maximum values.

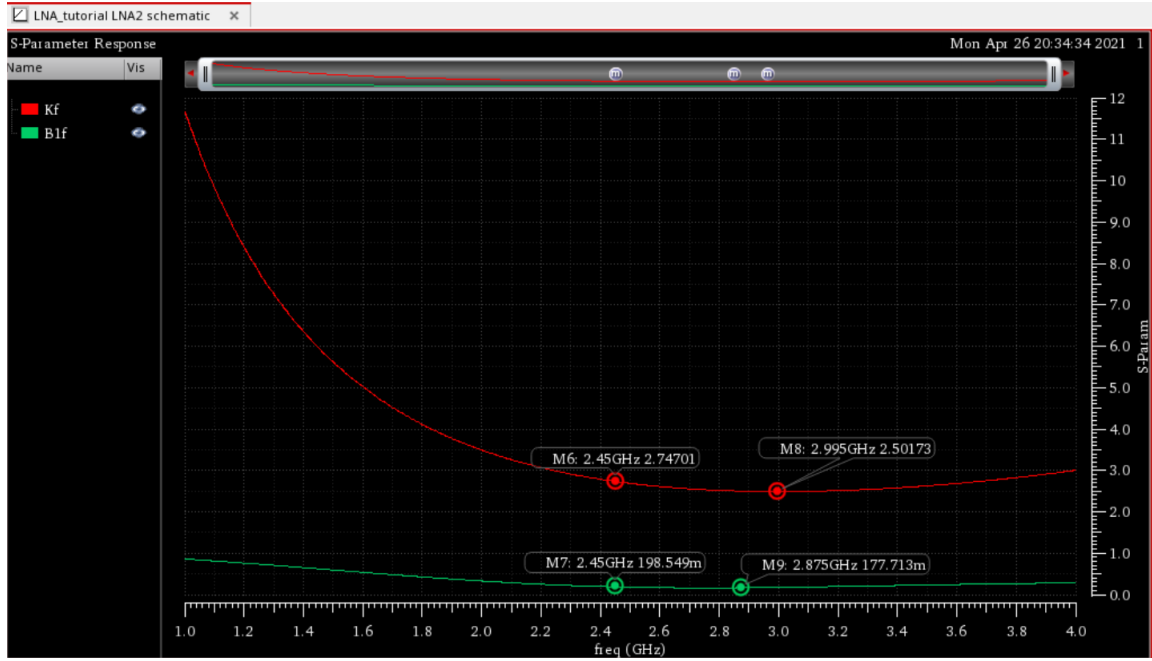


Figure 14. Stability parameters plots

Is the LNA unconditionally stable?

Yes, according to the conditions  $K_f > 1$  and  $B_{1f} > 0$ .

## 2.6. Evaluating Noise performance

Capture the plots of NF obtained, including markers at  $f_0 = 2.45$  GHz.

I've obtained the Noise Figure, NF, from ADE - Results - Direct Plot - Noise Figure, without using the Direct Plot.

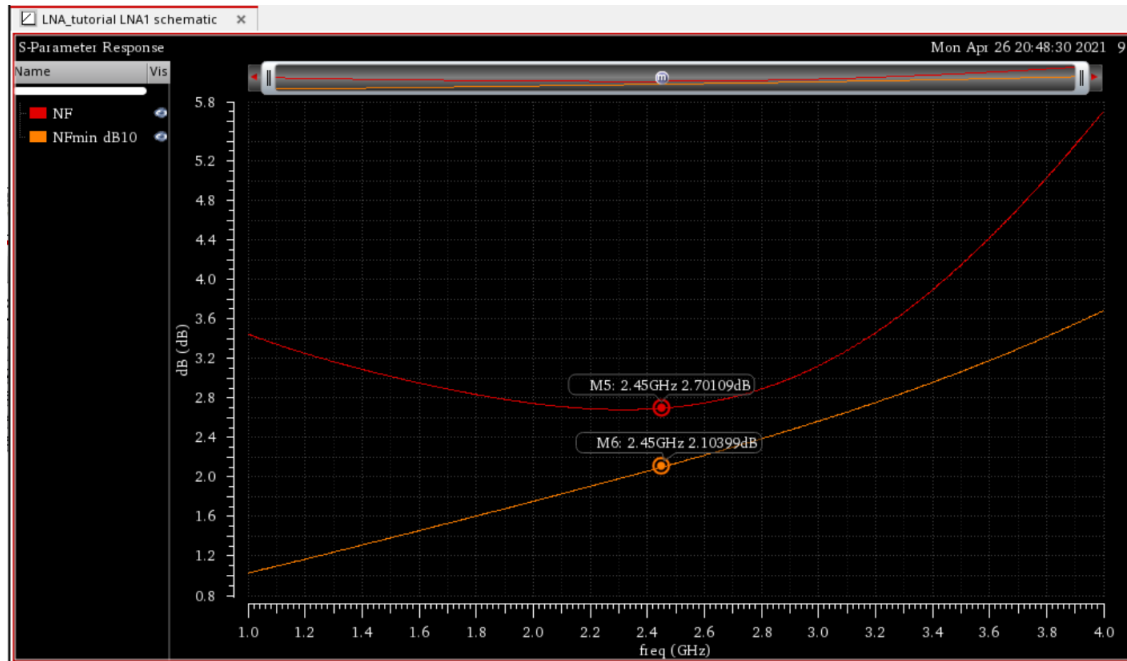


Figure 15. NF plots

**Is the NF close to the value theoretically expected?**

The theoretical value, with the professor's numbers, is

$$NF = 2.6 = 4.15 \text{ dB} . \quad (45)$$

This doesn't fulfill the desired NF,  $NF < 3 \text{ dB}$ , but in the pre-lab we were told we could fix this by increasing the current consumption to increase  $f_T$ .

However, after the simulation, the obtained NF is better than the theoretical one, and it fulfills  $NF < 3 \text{ dB}$ .

**Does the LNA fulfill the targeted NF specification?**

As commented, yes. As commented in the lab, this analysis tends to be optimistic, as it is done after linearizing the equations. If a large-signal analysis was performed, NF would tend to be worse (larger).

**Capture the (upper part of the) Noise Summary window.**

Device	Param	Noise Contribution	% Of Total
/PORT1	rn	3.98999e-16	53.69
L2.Rs	rn	9.33806e-17	12.57
MN1.rg	rn	7.9436e-17	10.69
/MN1	id	6.43028e-17	8.65
L2.Rsub2	rn	3.47731e-17	4.68
/MN2	id	2.4622e-17	3.31
L0.Rs	rn	1.30886e-17	1.76
MN1.rsource	rn	8.27166e-18	1.11
L1.Rs	rn	7.71368e-18	1.04
MN1.rsub1	rn	4.09153e-18	0.55
L2.Rsub1	rn	3.89345e-18	0.52
MN2.rsub1	rn	2.4023e-18	0.32
MN2.rg	rn	2.33658e-18	0.31
L0.Rsub2	rn	2.04864e-18	0.28
L1.Rsub1	rn	1.79778e-18	0.24
MN2.rsource	rn	8.76044e-19	0.12
MN1.rdrain	rn	4.84123e-19	0.07
/R0	rn	4.06738e-19	0.05
MN2.rdrain	rn	1.01405e-19	0.01
/MN1	fn	7.14579e-20	0.01
MN1.rsub2	rn	3.14635e-20	0.00
/MN2	fn	2.33005e-20	0.00
MN2.rsub2	rn	3.33227e-21	0.00
/MN5	id	2.45583e-21	0.00
MN5.rg	rn	3.32505e-22	0.00
MN5.rsource	rn	1.34055e-22	0.00
MN5.rsub1	rn	3.93089e-23	0.00
MN5.rdrain	rn	3.7449e-24	0.00
/MN5	fn	1.90711e-24	0.00
MN5.rsub2	rn	7.32871e-26	0.00
MN1.djdb	id	1.436e-29	0.00
MN2.djdb	id	4.9677e-30	0.00
MN1.djsb	id	5.08718e-33	0.00

Figure 16. Noise summary

Identify the noise sources that contribute more than 5% to the total noise.

As can be seen from the figure, only 4 noise sources contribute more than 5% to the total noise. It's required to understand their origin.

*/PORT1* : Input voltage source 50  $\Omega$  equivalent resistance

*L2.Rs* : Gate inductance series resistor

*MN1.rg* : Input NMOS equivalent gate resistance

*MN1* : Input NMOS channel noise

(46)

What does it mean that noise of PORT1 contributes more than 50%?

That the majority of the noise at the output comes from the input equivalent circuit, and thus that the LNA noise contribution is lower than the source one. The  $NF = 2.7$  dB is equal to 1.365 in linear units. Recall

$$F = 1 + \frac{R_{Lg}}{R_s} + \frac{R_{Rg}}{R_s} + \frac{\gamma}{\alpha} g_m R_s \left( \frac{w_0}{w_T} \right)^2 . \quad (47)$$

And we normally used

$$F_{min} = 1 + \frac{R_{Lg}}{R_s} + \frac{R_{Rg}}{R_s} + 2.4 \frac{\gamma}{\alpha} \frac{w_0}{w_T} . \quad (48)$$

In our analysis, we neglected the NMOS equivalent gate resistance contribution. In the pre-lab, we had  $R_{Lg} = 16 \Omega$ . We calculated  $f_T = 11.2$  GHz and we were given  $\gamma = 2$ ,  $\alpha = 0.8$ . This lead us to  $F = 2.6$ .

**Is the contribution of the gate inductor  $L_G$  larger of smaller than that calculated?**

Now, we have

$$\frac{R_{Lg}}{R_s} = \frac{L_G R_s}{/PORT1} = 0.234 . \quad (49)$$

Before, with  $R_{Lg} = 16 \Omega$ ,  $\frac{R_{Lg}}{R_s} = 0.32$ . So, the gate inductor contribution is lower, now.

**Is the contribution of the channel thermal noise larger of smaller than that calculated?**

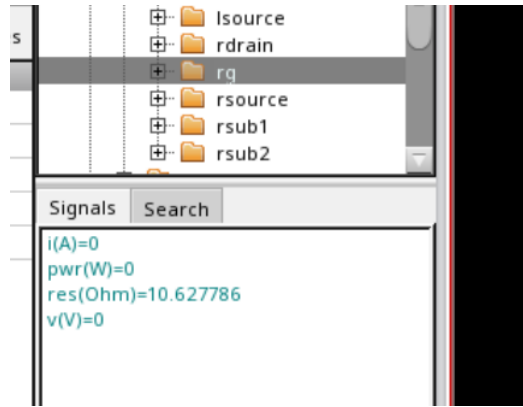
Now, we have

$$2.4 \frac{\gamma}{\alpha} \frac{w_0}{w_T} = \frac{MN1}{/PORT1} = 0.16 . \quad (50)$$

While before this term was  $\frac{\gamma}{\alpha} \frac{w_0}{w_T} = 1.267$ . So, the simulation attributes less noise to the channel than the one we calculated. It helps that  $w_T$  is greater than in the pre-lab calculations, as shown in a table before. The low channel thermal noise is the main source of noise improvement.

**Is the value of the gate resistance coherent with the contribution to the total output noise? Justify your answer**

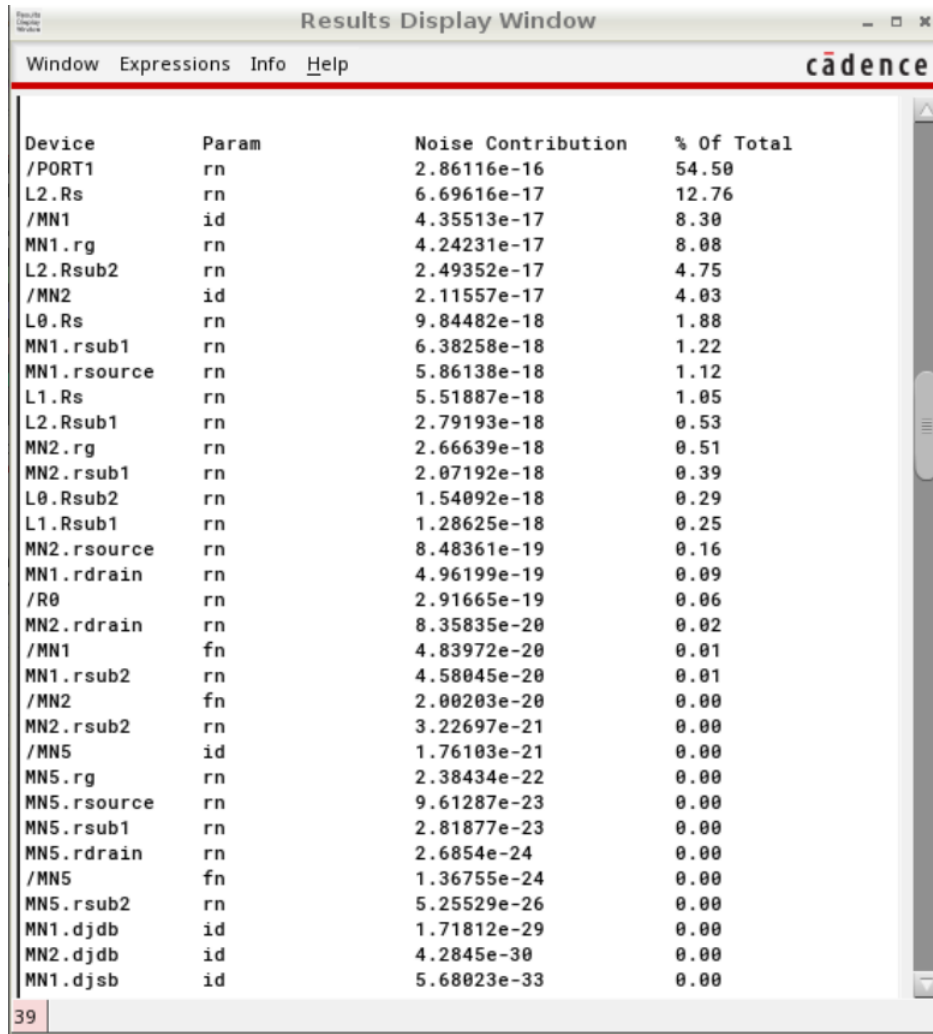
The gate resistance value I get is  $R_g = 10.63 \Omega$ .

Figure 17.  $R_g$  value at the CS transistor

So, it's contribution to the total noise should be  $\frac{R_g}{R_s} = \frac{10.63}{50} = 0.2126$ , normalized to the source noise (/PORT1). In the noise summary, the relation is  $\frac{10.69}{53.69} = 0.199$ . So, the value of the gate resistance is coherent with the contribution to the total output noise.

**Optional:** You can try to improve the NF by decreasing the gate resistance. We know this resistance can be reduced by layout, i.e. decreasing the finger size (and increasing the number of fingers, to preserve the total width).

I've decreased from 10 to 5 the stripe width, and instead of  $N$  now I've set  $2N$ . A small decrease of noise results.



The screenshot shows the 'Results Display Window' from Cadence. The window has a menu bar with 'Window', 'Expressions', 'Info', and 'Help'. The title bar says 'Results Display Window' and the Cadence logo is in the top right. The main area displays a table with the following data:

Device	Param	Noise Contribution	% Of Total
/PORT1	rn	2.86116e-16	54.50
L2.Rs	rn	6.69616e-17	12.76
/MN1	id	4.35513e-17	8.30
MN1.rg	rn	4.24231e-17	8.08
L2.Rsub2	rn	2.49352e-17	4.75
/MN2	id	2.11557e-17	4.03
L0.Rs	rn	9.84482e-18	1.88
MN1.rsub1	rn	6.38258e-18	1.22
MN1.rsource	rn	5.86138e-18	1.12
L1.Rs	rn	5.51887e-18	1.05
L2.Rsub1	rn	2.79193e-18	0.53
MN2.rg	rn	2.66639e-18	0.51
MN2.rsub1	rn	2.07192e-18	0.39
L0.Rsub2	rn	1.54092e-18	0.29
L1.Rsub1	rn	1.28625e-18	0.25
MN2.rsource	rn	8.48361e-19	0.16
MN1.rdrain	rn	4.96199e-19	0.09
/R0	rn	2.91665e-19	0.06
MN2.rdrain	rn	8.35835e-20	0.02
/MN1	fn	4.83972e-20	0.01
MN1.rsub2	rn	4.58045e-20	0.01
/MN2	fn	2.00203e-20	0.00
MN2.rsub2	rn	3.22697e-21	0.00
/MN5	id	1.76103e-21	0.00
MN5.rg	rn	2.38434e-22	0.00
MN5.rsource	rn	9.61287e-23	0.00
MN5.rsub1	rn	2.81877e-23	0.00
MN5.rdrain	rn	2.6854e-24	0.00
/MN5	fn	1.36755e-24	0.00
MN5.rsub2	rn	5.25529e-26	0.00
MN1.djdb	id	1.71812e-29	0.00
MN2.djdb	id	4.2845e-30	0.00
MN1.djsb	id	5.68023e-33	0.00

The status bar at the bottom left shows the page number '39'.

Figure 18. Noise summary

Now the contribution from the gate resistance is the previous noise by a factor of  $\approx 0.534$ , so the improvement is noticeable. Still, in both cases the contribution is quite small overall, so there's not a huge impact on the total noise.

## 2.7. Evaluating Linearity

Capture the plot generated for the measurement of IP<sub>3</sub>.

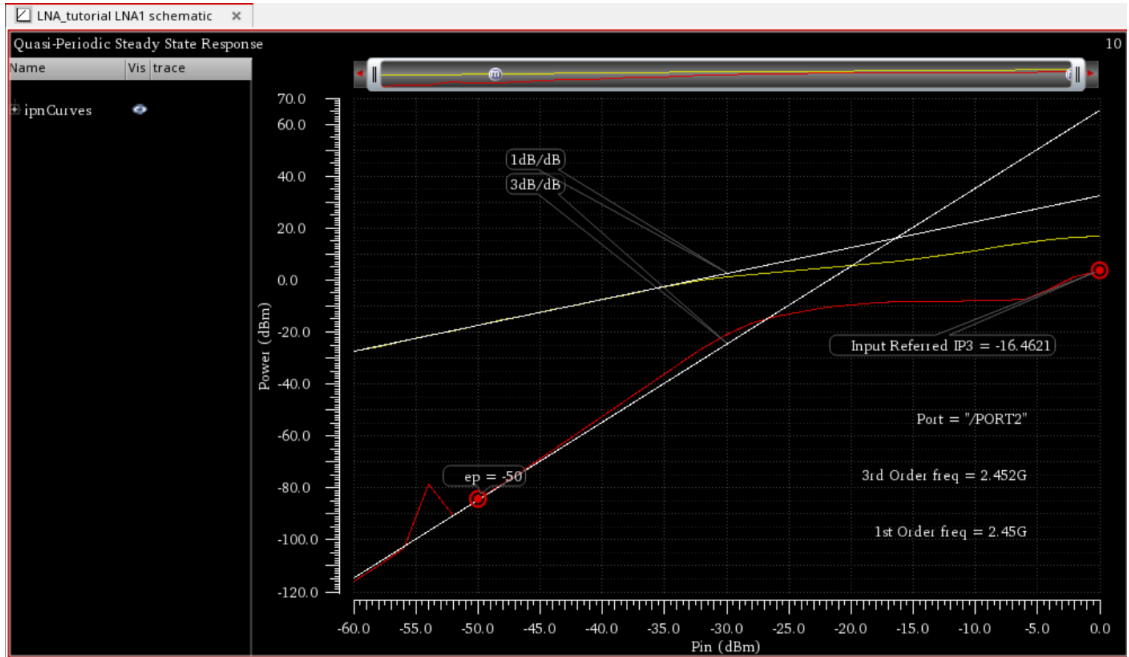


Figure 19. IP3 analysis

Which is the input-referred IP3 of your LNA?

I get

$$IIP3 = -16.4621 \text{ dBm} . \quad (51)$$

Which is the OIP3?

Taking the previous  $A_v = 33 \text{ dB}$ , I would have

$$OIP3 = 16.54 \text{ dBm} . \quad (52)$$

This value can be, more or less, deduced on the previous plot, in the vertical axis.

Which other frequencies you could have selected for the fundamental tone and for the 3rd-order intermodulation product?

The chosen frequencies,  $f_1 = 2.45 \text{ GHz}$  and  $f_2 = 2.451 \text{ GHz}$  result in third-order tones at 2.452 GHz and 2.448 GHz, and thus I've chosen 2.45 GHz as the 1<sup>st</sup> order harmonic, and 2.452 GHz as the 3<sup>rd</sup> order harmonic. I could have also chosen 2.451 GHz as the 1<sup>st</sup> order harmonic, and 2.449 GHz as the 3<sup>rd</sup> order harmonic, because the two input tones are of the same power, and the tone at 2.449 GHz will be equal to the one at 2.451 GHz. I've noticed that for this, the results were almost identical.

Capture the plot generated for the measurement of CP1dB.



Figure 20. Compression analysis

Which is the input-referred 1dB Compression point of your LNA?

I get

$$P_{in,-1dB} = -31.636 \text{ dBm} . \quad (53)$$

Which is peak input amplitude that corresponds to this CP1dB?

By recalling that

$$P[dBm] = 10 \log \left( \frac{(V_{peak}/\sqrt{2})^2}{50} \right) , \quad (54)$$

I get

$$V_{in,peak,-1dB} = 8.283 \text{ mV}_{peak} . \quad (55)$$

Which is the output-referred 1dB Compression point of your LNA?

$$P_{out,-1dB} = 1.364 \text{ dBm} . \quad (56)$$



Which is peak output amplitude that corresponds to this CP1dB?

A similar calculation to the one before is performed. I get

$$V_{out,peak,-1dB} = 0.370 V_{peak} . \quad (57)$$

**Optional:** it is interesting that you repeat the 1dB compression point representation, but now selecting Power Gain as the output format representation. This provides an alternative interpretation of the 1dB compression point. At small input power, the gain is constant, and the 1dB compression point is the power at which the gain is reduced 1dB.

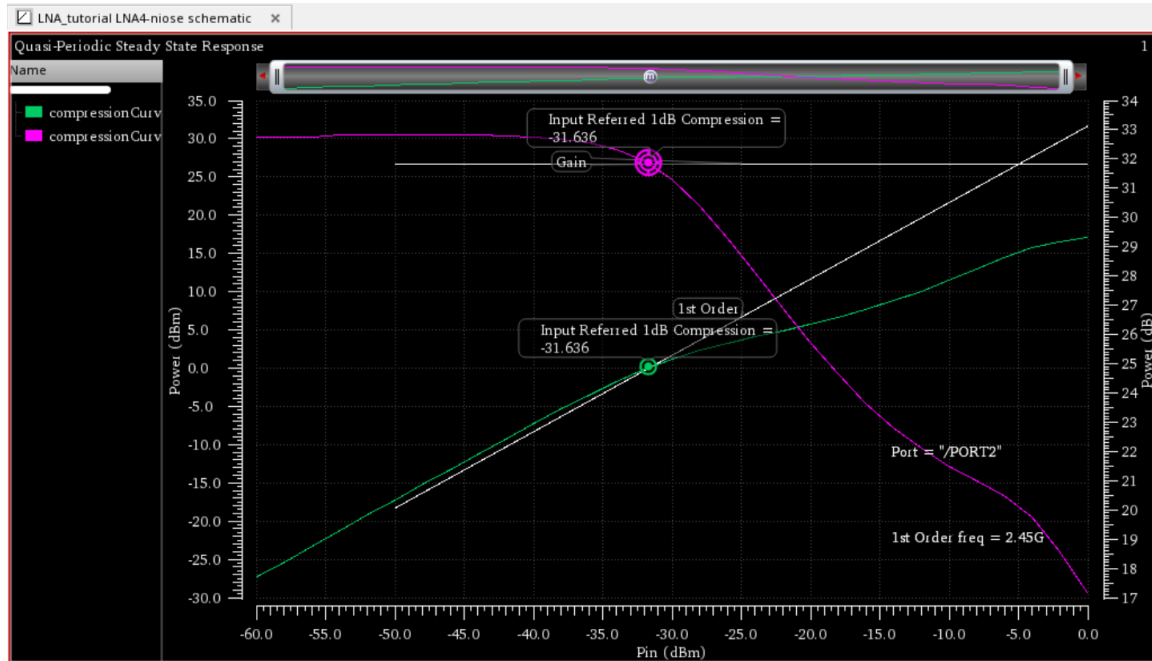


Figure 21. Compression analysis

I have done it and the resulting plot is more or less as the statement says. For low input powers the response is more or less flat, and for high inputs compression starts taking place and the gain diminishes. The marked compression point is the same as before.

## 2.8. Performance Summary

Report the final performance results obtained, fulfilling the next table:

I've obtained different  $S_{21}$  values during this lab. Some were obtained when there were mismatches, or when the buffer was deleted. We are told "LNA output voltage will drive the input of an on-chip mixer, thus LNA output impedance is not intended to be matched", so it makes

more sense to take the first  $S_{21}$  I obtained.

Characteristic	Value
Central frequency $f_0$	2.45 GHz
$S_{21}$ (dB)	32.9953
NF	2.70109 dB
IIP3 (dBm)	-16.4621
Input - CP 1dB (dBm)	-31.636
$S_{11}$ (dB)	-9.80
Reverse Isolation, $S_{12}$ (dB)	-37.435
VDD	3.3 V
IDC (mA)	2.629
Power Dissipation (mW)	8.6757
Technology Process	C35B4M3, 0.35 $\mu\text{m}$

Table 4. Final performance results

**Optional:** You may want to compare the LNA performance you obtained against the performance of LNA designed by international research groups, published in research journals or congresses. A fair comparison would restrict to LNAs designed in similar technology nodes, similar central frequencies, single-ended topologies. We propose eg. the following works:

Characteristic	LNA lab	Song	Shaeffer
Central frequency $f_0$	2.45 GHz	2.1 GHz	1.5 GHz
$S_{21}$ (dB)	32.9953	16.4	22
NF	2.70109 dB	2.77 dB	3.5 dB
IIP3 (dBm)	-16.4621	7.45	-9.3
Input - CP 1dB (dBm)	-31.636	-	-22
$S_{11}$ (dB)	-9.80	-19.7	1.385 (?)
Reverse Isolation, $S_{12}$ (dB)	-37.435	-33.4	-42.451
VDD	3.3 V	1.8 V	1.5 V
IDC (mA)	2.629	13.88	20
Power Dissipation (mW)	8.6757	25	30
Technology Process	C35B4M3, 0.35 $\mu\text{m}$	0.25 $\mu\text{m}$	0.6 $\mu\text{m}$

Table 5. Comparison table

**Optional:** Performance comparison to other circuits may be difficult if they operate at different central frequencies. Also, different performance can be

traded-off, thus one metric (eg gain) can be better than your work, while other (eg. linearity) can be worse. To ease comparison, Figures of Merit (FoM) have been defined, that ease comparison.

We propose you compare your LNA against the designed published in other works, in terms of the following FoM:

$$FoM_1(GHz) = \frac{G_p(lin) \cdot f_0(GHz) \cdot IIP3(mW)}{(F(lin) - 1) \cdot Power(mW)} . \quad (58)$$

I guess  $G_p(lin)$  is the  $S_{21}$  in linear units. Some intermediate calculations:

$$S_{21} \text{ linear} = 44.61, 6.6069, 12.589$$

$$NF \text{ linear} = 1.86, 1.89, 2.239 \quad (59)$$

$$IIP3 \text{ [mW]} = 0.02258, 5.559, 0.1175$$

I get

Characteristic	LNA lab	Song	Shaeffer
$FoM_1$ (GHz)	0.3307 GHz	3.466 GHz	0.05969 GHz

Table 6. Comparison table

To my surprise, I've been able to get a better figure of merit than the one that from Shaeffer results. One reason may be that Shaeffer paper is quite old, and that I may be using a better technology. The other is that this figure of merit considers a lot of variables, and that the comparison may not be totally fair.

## 3. Code

### 3.1. VHDL

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  -----
7  -----
8  --
9  -- adc_driver.vhd
10 -- Hardware controller for the A/D converter
11 -- Provides the digitized input serial signal in a 8 bit output, although the 12 useful
    bits are stored
12 -- Clock frequency required: 108 MHz
13 -- resetn: active low
14 --
15 -----
16 -----
17
18
19 entity adc_driver is
20 port(
21     clk:      in std_logic; -- 108 MHz clock signal
22     resetn: in std_logic; -- active low
23     sdata1: in std_logic; -- serial data 1, comes from the ADC board, it's the one that
        provides signal
24     sdata2: in std_logic; -- serial data 2, comes from the ADC board
25     ncs: out std_logic; -- chip select signal for the ADC converters
26     sclk: out std_logic; -- slow clock, 108 MHz / 6 = 18 MHz clock for the ADC converters
27     sample_ready: out std_logic; -- turns 1 for a clock period to indicate new data is
        present at the output
28     data: out std_logic_vector(11 downto 0) -- output data
29 );
30 end adc_driver;
31
32 architecture arch of adc_driver is
33     type states is (start, t2, readzeros, readbits, tquiet); -- possible states in the
        conversion
34     signal state: states := start; -- initialize state
35     signal counter: std_logic_vector(2 downto 0); -- counts the number of clk periods to
        know when it's time to jump to another state
36     signal counter_sclk: std_logic_vector(2 downto 0); -- will count from 0 to 5 in order
        to generate the slow clock, sclk
37     signal counter_bits: std_logic_vector(3 downto 0); -- counts the first 3 useless bits
        , return to 0, and then counts the 12 useful bits
38     signal dataaa: std_logic_vector(11 downto 0); -- internal data
39     signal sample_readyy: std_logic; -- internal sample_ready signal
40 begin
41
42
43 main: process (clk) -- driver
44 begin
45     if (clk'event and clk='1') then
46         if (resetn='0') then
47             sclk <= '1';
48             ncs <= '1';
49             counter_bits <= (others => '0');
50             sample_readyy <= '0';
51         else
52             case (state) is
53                 when start => -- set chip select to 0 and move to next state
54                     state <= t2;

```

```

55         ncs <= '0';
56         counter <= (others => '0');
57         when t2 =>
58             acquiring
59             counter <= counter + 1;
60             if (counter = 1) then -- waits for 2 clk periods,
61                 around 20 ns
62                 state <= readzeros;
63                 counter <= (others => '0');
64                 sclk <= '0';
65                 counter_sclk <= (others => '0');
66             end if;
67             when readzeros =>
68                 confusing
69                 counter_sclk <= counter_sclk + 1;
70                 if (counter_sclk = 2) then -- sclk rising edge
71                     sclk <= '1';
72                 elsif (counter_sclk = 5) then
73                     sclk <= '0'; -- sclk falling edge
74                     counter_sclk <= (others => '0');
75                     counter_bits <= counter_bits + 1;
76                     if (counter_bits = 3-1) then -- the 3 Z bits have
77                         been received
78                         counter_bits <= (others => '0');
79                         state <= readbits;
80                     end if;
81                 end if;
82                 when readbits =>
83                     -- read the 12 useful bits
84                     counter_sclk <= counter_sclk + 1;
85                     if (counter_sclk = 2) then -- sclk rising edge
86                         sclk <= '1';
87                     elsif (counter_sclk = 5) then -- sclk falling edge
88                         sclk <= '0';
89                         counter_sclk <= (others => '0');
90                         counter_bits <= counter_bits + 1;
91                         dataa(11 - to_integer(unsigned(counter_bits))) <=
92                             sdata1; -- msb comes first
93                         if (counter_bits = 12-1) then -- all 12 bits
94                             received
95                             counter_bits <= (others => '0');
96                             state <= tquiet;
97                             sample_readyy <= '1';
98                             counter <= (others => '0');
99                         end if;
100                     end if;
101                     when tquiet =>
102                         -- wait for enough time until starting the next
103                         conversion
104                         sample_readyy <= '0'; -- 1 period width
105                         counter_sclk <= counter_sclk + 1;
106                         if (counter_sclk = 2) then -- return sclk and ncs to
107                             idle (1)
108                             sclk <= '1';
109                             ncs <= '1';
110                         end if;
111                         if (counter_sclk = 7) then -- during 5 clk periods,
112                             around 50 ns = tquiet, sclk and ncs have remained
113                             idle
114                             state <= start;
115                             counter <= (others => '0');
116                         end if;
117                     end case;
118                 end if;
119             end if;
120         end process;
121 output: process (clk) -- output data register

```

```

112 begin
113     if (clk'event and clk='1') then
114         if (resetsn = '0') then
115             data <= (others => '0');
116             sample_ready <= '0';
117         else
118             if (sample_readyy = '1') then
119                 data <= dataaa;
120             end if;
121             sample_ready <= sample_readyy;
122         end if;
123     end if;
124 end process;
125
126 end arch;

```

### 3.2. C

```

1  /*
2  ****
3  *                                     EXAMPLE CODE
4  *
5  *                                     (c) Copyright 2009-2015; Micrium, Inc.; Weston, FL
6  *
7  *                                     All rights reserved.  Protected by international copyright laws.
8  *
9  *                                     Please feel free to use any application code labeled as 'EXAMPLE CODE' in
10 *                                     your application products.  Example code may be used as is, in whole or
11 *                                     in
12 *                                     part, or may be used as a reference only.
13 *
14 *                                     Please help us continue to provide the Embedded community with the finest
15 *                                     software available.  Your honesty is greatly appreciated.
16 *
17 *                                     You can contact us at www.micrium.com.
18 ****
19 */
20
21 /*
22 ****
23 *                                     SETUP INSTRUCTIONS
24 *
25 *                                     This demonstration project illustrate a basic uC/OS-III project with simple "hello
26 *                                     world" output.
27 *
28 *                                     By default some configuration steps are required to compile this example :
29 *
30 *                                     1. Include the require Micrium software components
31 *                                     In the BSP setting dialog in the "overview" section of the left pane the
32 *                                     following libraries
33 *                                     should be added to the BSP :
34 *
35 *                                     ucos_common
36 *                                     ucos_osiii
37 *                                     ucos_standalone
38 *
39 *                                     2. Kernel tick source - (Not required on the Zynq-7000 PS)
40 *                                     If a suitable timer is available in your FPGA design it can be used as the kernel
41 *                                     tick source.
42 *                                     To do so, in the "ucos" section select a timer for the "kernel_tick_src"
43 *                                     configuration option.
44 *
45 *                                     3. STDOUT configuration
46 *                                     Output from the print() and UCOS_Print() functions can be redirected to a
47 *                                     supported UART. In

```

```

42 *      the "ucos" section the stdout configuration will list the available UARTs.
43 *
44 *      Troubleshooting :
45 *      By default the Xilinx SDK may not have selected the Micrium drivers for the timer
46 *      and UART.
47 *      If that is the case they must be manually selected in the drivers configuration
48 *      section.
49 *
50 *      Finally make sure the FPGA is programmed before debugging.
51 *
52 *      Remember that this example is provided for evaluation purposes only. Commercial
53 *      development requires
54 *      a valid license from Micrium.
55 ****
56 */
57 /*
58 ****
59 *                                     INCLUDE FILES
60 ****
61 */
62
63 #include <stdio.h>
64 #include <Source/os.h>
65 #include <ucos_bsp.h>
66
67 #include "xadcps.h"
68 #include <xgpio.h>
69
70
71 /*
72 ****
73 *                                     DEFINES
74 ****
75 */
76
77 #define XADC_DEVICE_ID      XPAR_XADCPS_0_DEVICE_ID
78 #define GPIO_DEVICE_ID     XPAR_AXI_GPIO_0_DEVICE_ID
79 #define BUTTON_CHANNEL     1 // Input channel of the GPIO (check this is consistent with
80 *                           the block diagram)
81 #define TEMPERATURE_CHANNEL 2 // Output channel of the GPIO (check this is consistent
82 *                           with the block diagram)
83 #define APP_TASK_START_STK_SIZE 512u
84 #define APP_TASK1_STK_SIZE      512u
85 #define APP_TASK2_STK_SIZE      512u
86 #define APP_TASK_START_PRIO     8u
87 #define APP_TASK1_PRIO          2u
88 #define APP_TASK2_PRIO          3u
89
90 /*
91 ****
92 *                                     LOCAL VARIABLES
93 ****
94 */
95
96 static OS_TCB      AppTaskStartTCB; // Task Control Block (
97 *                           TCB).
98 static OS_TCB      AppTask1TCB;
99 static OS_TCB      AppTask2TCB;
100
101 static CPU_STK      AppTaskStartStk[APP_TASK_START_STK_SIZE]; // Startup Task Stack
102 static CPU_STK      AppTask1Stk[APP_TASK1_STK_SIZE]; // Task #1 Stack
103 static CPU_STK      AppTask2Stk[APP_TASK2_STK_SIZE]; // Task #2 Stack

```

```

103 static OS_MUTEX      AppMutexPrint;                // App Mutex
104
105 static XAdcPs XAdcInst;                             // XADC Driver instance
106 XAdcPs *XAdcInstPtr = &XAdcInst;
107 static XGpio Gpio;                                  // GPIO Driver instance
108
109 // Global variable that holds the output
110 int output = 0; // 32 bit variable, contains 23 useful bits. threshold(11b)|temperature
111                (11b)|alarm(1b)
112 int threshold; // temperature threshold
113 int temperature; // last read temperature value
114 int alarm; // holds 1 if there's an alarm (temperature > threshold)
115
116 /*
117 ****
118 *                                LOCAL FUNCTION PROTOTYPES
119 ****
120 */
121
122 static void AppTaskCreate      (void);
123 static void AppTaskStart      (void *p_arg);
124 static void AppTask1          (void *p_arg);
125 static void AppTask2          (void *p_arg);
126 static void AppPrintWelcomeMsg (void);
127 static void AppPrint          (char *str);
128 static void AppPrintWelcomeMsg (void);
129 static void Peripheral_Init    (void); //initialization of the peripheral unit for the
130                XadcPs
131 void MainTask (void *p_arg);
132
133 /*
134 ****
135 *                                main()
136 *
137 * Description : Entry point for C code.
138 *
139 ****
140 */
141 int main()
142 {
143     threshold = 50;                // Initialize threshold, for practicality purposes
144
145     UCOSStartup(MainTask);
146
147     return 0;
148 }
149
150 /*
151 ****
152 *                                STARTUP TASK
153 *
154 * Description : This is an example of a startup task.
155 *
156 * Arguments   : p_arg   is the argument passed to 'AppTaskStart()' by 'OSTaskCreate()'.
157 *
158 * Returns     : none
159 *
160 * Notes       :
161 ****
162 */
163 void MainTask (void *p_arg)
164 {
165     OS_ERR      err;
166
167     AppPrintWelcomeMsg();

```



```

168
169     OSInit(&err);          /* Initialize uC/OS-III.                      */
170
171     OSTaskCreate    ((OS_TCB    *)&AppTaskStartTCB,
172                     (CPU_CHAR   *)&"App Task Start",
173                     (OS_TASK_PTR )AppTaskStart,
174                     (void       *)&0,
175                     (OS_PRIO     )APP_TASK_START_PRIO,
176                     (CPU_STK     *)&AppTaskStartStk[0],
177                     (CPU_STK_SIZE)APP_TASK_START_STK_SIZE / 10,
178                     (CPU_STK_SIZE)APP_TASK_START_STK_SIZE,
179                     (OS_MSG_QTY  )0,
180                     (OS_TICK     )0,
181                     (void       *)&0,
182                     (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
183                     (OS_ERR *)&err);
184
185     OSStart(&err);          /* Start multitasking (i.e. give control to uC/OS-II). */
186
187     while (1) {
188         AppPrint(".");
189         ;
190     }
191 }
192
193 /*
194 *****
195 *
196 *                               PRINT WELCOME THROUGH UART
197 *
198 * Description : Prints a welcome message through the UART.
199 *
200 * Argument(s) : none
201 *
202 * Return(s)   : none
203 *
204 * Caller(s)   : application functions.
205 *
206 * Note(s)     : Because the welcome message gets displayed before
207 *               the multi-tasking has started, it is safe to access
208 *               the shared resource directly without any mutexes.
209 *****
210 */
211 static void AppPrintWelcomeMsg (void)
212 {
213     UCOS_Print("\f\f\r\n");
214     UCOS_Print("Micrium\r\n");
215     UCOS_Print("uCOS-III\r\n\r\n");
216     UCOS_Print("This application runs three different tasks:\r\n\r\n");
217     UCOS_Print("1. Task Start: Initializes the OS and creates tasks and\r\n");
218     UCOS_Print("    other kernel objects such as the mutex.\r\n");
219     UCOS_Print("    This task remains running and printing a\r\n");
220     UCOS_Print("    dot '.' every 100 milliseconds.\r\n");
221     UCOS_Print("2. Task #1 : Reads temperature every 200-milliseconds.\r\n");
222     UCOS_Print("3. Task #2 : Reads input buttons every 500-milliseconds.\r\n\r\n");
223 }
224
225 /*
226 *****
227 *
228 *                               STARTUP TASK
229 *
230 * Description : This is an example of a startup task. As mentioned in the book's text,
231 *               you MUST
232 *               initialize the ticker only once multitasking has started.
233 *
234 * Arguments   : p_arg is the argument passed to 'AppTaskStart()' by 'OSTaskCreate()'.
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
834 *
835 *
836 *
837 *
838 *
839 *
840 *
841 *
842 *
843 *
844 *
845 *
846 *
847 *
848 *
849 *
850 *
851 *
852 *
853 *
854 *
855 *
856 *
857 *
858 *
859 *
860 *
861 *
862 *
863 *
864 *
865 *
866 *
867 *
868 *
869 *
870 *
871 *
872 *
873 *
874 *
875 *
876 *
877 *
878 *
879 *
880 *
881 *
882 *
883 *
884 *
885 *
886 *
887 *
888 *
889 *
890 *
891 *
892 *
893 *
894 *
895 *
896 *
897 *
898 *
899 *
900 *
901 *
902 *
903 *
904 *
905 *
906 *
907 *
908 *
909 *
910 *
911 *
912 *
913 *
914 *
915 *
916 *
917 *
918 *
919 *
920 *
921 *
922 *
923 *
924 *
925 *
926 *
927 *
928 *
929 *
930 *
931 *
932 *
933 *
934 *
935 *
936 *
937 *
938 *
939 *
940 *
941 *
942 *
943 *
944 *
945 *
946 *
947 *
948 *
949 *
950 *
951 *
952 *
953 *
954 *
955 *
956 *
957 *
958 *
959 *
960 *
961 *
962 *
963 *
964 *
965 *
966 *
967 *
968 *
969 *
970 *
971 *
972 *
973 *
974 *
975 *
976 *
977 *
978 *
979 *
980 *
981 *
982 *
983 *
984 *
985 *
986 *
987 *
988 *
989 *
990 *
991 *
992 *
993 *
994 *
995 *
996 *
997 *
998 *
999 *
1000 *

```

```

234 * Returns      : none
235 *
236 * Notes        : 1) The first line of code is used to prevent a compiler warning because '
      p_arg' is not
237 *              used. The compiler should not generate any code for this statement.
238 *****
239 */
240
241 static void AppTaskStart (void *p_arg)
242 {
243     OS_ERR    err;
244
245     UCOS_Print("Task Start Created\r\n");
246
247     AppTaskCreate();                                /* Create Application
      tasks                                           */
248     OSMutexCreate((OS_MUTEX *)&AppMutexPrint, (CPU_CHAR *)"My App. Mutex", (OS_ERR *)&err
      );
249
250     while (1) {                                    /* Task body, always written
      as an infinite loop.                            */
251
252         OSTimeDlyHMSM(0, 0, 0, 100,
253                     OS_OPT_TIME_HMSM_STRICT,
254                     &err);                          /* Waits 100 milliseconds
      .                                                */
255
256         AppPrint(".");                               /* Prints a dot every 100
      milliseconds.                                  */
257     }
258 }
259
260 /*
261 *****
262 *              CREATE APPLICATION TASKS
263 *
264 * Description : Creates the application tasks.
265 *
266 * Argument(s) : none
267 *
268 * Return(s)   : none
269 *
270 * Caller(s)   : AppTaskStart()
271 *
272 * Note(s)     : none.
273 *****
274 */
275
276 static void AppTaskCreate (void)
277 {
278     OS_ERR    err;
279
280
281     OSTaskCreate((OS_TCB *)&AppTask1TCB,                                /* Create the Task #1.
      (CPU_CHAR *) "Task 1",
282     (OS_TASK_PTR ) AppTask1,
283     (void *) 0,
284     (OS_PRIO ) APP_TASK1_PRIO,
285     (CPU_STK *) &AppTask1Stk[0],
286     (CPU_STK_SIZE) APP_TASK1_STK_SIZE / 10u,
287     (CPU_STK_SIZE) APP_TASK1_STK_SIZE,
288     (OS_MSG_QTY ) 0u,
289     (OS_TICK ) 0u,
290     (void *) 0,
291     (OS_OPT ) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
292     (OS_ERR *) &err);
293

```

```

294
295     OSTaskCreate((OS_TCB      *)&AppTask2TCB,                      /* Create the Task #2.
296                                     */
297                 (CPU_CHAR    *)"Task 2",
298                 (OS_TASK_PTR ) AppTask2,
299                 (void        *) 0,
300                 (OS_PRIO     ) APP_TASK2_PRIO,
301                 (CPU_STK     *)&AppTask2Stk[0],
302                 (CPU_STK_SIZE) APP_TASK2_STK_SIZE / 10u,
303                 (OS_MSG_QTY  ) 0u,
304                 (OS_TICK     ) 0u,
305                 (void        *) 0,
306                 (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
307                 (OS_ERR      *)&err);
308 }
309
310 /*
311 *****
312 *                                     TASK #1
313 *
314 * Description : This is an example of an application task that prints "1" every second to
315 *               the UART.
316 *
317 * Arguments   : p_arg   is the argument passed to 'AppTaskStart()' by 'OSTaskCreate()'.
318 *
319 * Returns     : none
320 *
321 * Notes       : 1) The first line of code is used to prevent a compiler warning because '
322 *               p_arg' is not
323 *               used. The compiler should not generate any code for this statement.
324 *****
325 */
326 static void AppTask1 (void *p_arg) // Temperature task
327 {
328     OS_ERR  err;
329
330     unsigned int temperature_raw;
331     (void)p_arg;
332     char temp_string[20]; // Holds the temperature, it is later printed
333     char temp_string_pixels[20]; // Holds the temperature, it is later printed
334     char t_temp_string[20]; // Holds the temperature threshold, it is later printed
335     char t_temp_string_pixels[20]; // Holds the temperature threshold, it is later
336     printed
337     char alarm_string[20];
338
339     AppPrint("Temperature task has started\r\n");
340     Peripheral_Init();
341     while (1) {
342
343                                     /* Task body, always written as an
344                                     infinite loop. */
345
346         temperature_raw = XAdcPs_GetAdcData(XAdcInstPtr, XADCPS_CH_TEMP);
347         temperature = (int) XAdcPs_RawToTemperature(temperature_raw);
348
349         // Print read temperature
350         sprintf(temp_string, "%d", temperature);
351         AppPrint("\n Temperature: ");
352         AppPrint(temp_string);
353
354         sprintf(temp_string_pixels, "%u", temperature<<4);
355         AppPrint("\n Temperature pixels: ");
356         AppPrint(temp_string_pixels);
357
358         // Print threshold

```

```

356     sprintf(t_temp_string, "%d", threshold & 0x7F);
357     AppPrint("\n Threshold: ");
358     AppPrint(t_temp_string);
359
360     // Print threshold
361     sprintf(t_temp_string_pixels, "%u", (threshold & 0x7F)<<4);
362     AppPrint("\n Threshold pixels: ");
363     AppPrint(t_temp_string_pixels);
364
365     // Compare temperature against threshold
366     if (temperature > threshold){
367         alarm = 1;
368     }
369     else {
370         alarm = 0;
371     }
372
373     // Print alarm
374     sprintf(alarm_string, "%d", alarm);
375     AppPrint("\n alarm: ");
376     AppPrint(alarm_string);
377
378
379     output = (alarm & 0x1)<<22|(temperature & 0x7F)<<(4+11)|(threshold & 0x7F)<<(4); //
        Concatenate the variables to get 23 useful bits to output port
380
381
382     OSTimeDlyHMSM(0, 0, 0, 200,
383                   OS_OPT_TIME_HMSM_STRICT,
384                   &err);
385     // AppPrint("1");
386
387     XGpio_DiscreteWrite(&Gpio,TEMPERATURE_CHANNEL,output);    // Write the output in
        the gpio output channel
388
389
390 }
391 }
392
393
394
395 /*
396 ****
397 *                                     TASK #2
398 *
399 * Description : This is an example of an application task that prints "2" every 2 seconds
        to the UART.
400 *
401 * Arguments   : p_arg   is the argument passed to 'AppTaskStart()' by 'OSTaskCreate()'.
402 *
403 * Returns     : none
404 *
405 * Notes       : 1) The first line of code is used to prevent a compiler warning because '
        p_arg' is not
406 *               used. The compiler should not generate any code for this statement.
407 ****
408 */
409
410 static void AppTask2 (void *p_arg) // This is the responsible for the buttons
411 {
412     OS_ERR    err;
413
414
415     (void)p_arg;
416     int button=0;
417
418

```

```

419     AppPrint("Buttons task has started \r\n");
420     Peripheral_Init();
421     while (1) {
422
423         button = XGpio_DiscreteRead(&Gpio, BUTTON_CHANNEL); // Reads the input channel of the
424                                                                gpio to determine if a button has been pressed
425
426         // // Display read button value (0 if any button is pressed)
427         // char button_string[20];
428         // AppPrint("\n BUT ");
429         // sprintf(button_string, "%d", button);
430         // AppPrint(button_string);
431
432         if(button == 1) { // BTNL button pressed, decrease threshold by 1C
433             threshold = threshold-1;
434             AppPrint("Threshold - 1 \r\n");
435         }
436         else if (button == 2){ // BTNR button pressed, increase threshold by 1C
437             threshold = threshold+1;
438             AppPrint("Threshold + 1 \r\n");
439         }
440
441         output = (alarm & 0x1)<<22|(temperature & 0x7F)<<(4+11)|(threshold & 0x7F)<<(4);
442         // Concatenate the variables to get 23 useful bits to output port
443
444         XGpio_DiscreteWrite(&Gpio,TEMPERATURE_CHANNEL,output); // write in the gpio output
445                                                                channel
446
447                                                                /* Task body, always written as an
448                                                                infinite loop.          */
449
450         OSTimeDlyHMSM(0, 0, 0, 500,
451                     OS_OPT_TIME_HMSM_STRICT,
452                     &err);
453                                                                /* Waits for 2 seconds.
454                                                                */
455
456         // AppPrint("2");
457         UART.
458                                                                /* Prints 2 to the
459                                                                */
460     }
461 }
462
463 /*
464 *****
465 *
466 * PRINT THROUGH UART
467 *
468 * Description : Prints a string through the UART. It makes use of a mutex to
469 *               access this shared resource.
470 *
471 * Argument(s) : none
472 *
473 * Return(s)   : none
474 *
475 * Caller(s)   : application functions.
476 *
477 * Note(s)     : none.
478 *****
479 */
480
481 static void AppPrint (char *str)
482 {
483     OS_ERR err;

```

```

480     CPU_TS    ts;
481
482
483                                     /* Wait for the shared
                                     resource to be
                                     released.          */
484     OSMutexPend(    (OS_MUTEX *)&AppMutexPrint,
485                     (OS_TICK )0u,
486                     timeout,
487                     (OS_OPT )OS_OPT_PEND_BLOCKING,
488                     not available,
489                     (CPU_TS *)&ts,
490                     Timestamp,
491                     (OS_ERR *)&err);
492
493                                     /* Access the shared
494                                     resource.          */
495
496                                     /* Releases the shared
497                                     resource.          */
498
499                                     */
500     OSMutexPost(    (OS_MUTEX *)&AppMutexPrint,
501                     (OS_OPT )OS_OPT_POST_NONE,
502                     .,
503                     (OS_ERR *)&err);
504                                     /* No options
505                                     */
506 }
507
508 void Peripheral_Init()
509 {
510     int Status;
511     XAdcPs_Config *ConfigPtr;
512
513     /* Initialize the GPIO driver. If an error occurs then exit */
514     Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
515     if (Status != XST_SUCCESS) {
516         return XST_FAILURE;
517     }
518
519     /*
520     * Perform a self-test on the GPIO. This is a minimal test and only
521     * verifies that there is not any bus error when reading the data
522     * register
523     */
524     XGpio_SelfTest(&Gpio);
525
526     /*
527     * Setup direction register so the switch is an input and the LED is
528     * an output of the GPIO
529     */
530     XGpio_SetDataDirection(&Gpio, BUTTON_CHANNEL, 0xff); // Establish BUTTON_CHANNEL
531                                                             as an input
532
533     XGpio_SetDataDirection(&Gpio, TEMPERATURE_CHANNEL, 0x00); // Establish
534                                                             TEMPERATURE_CHANNEL as an output
535
536     /*
537     * Initialize the XAdc driver.
538     */
539     ConfigPtr = XAdcPs_LookupConfig(XADC_DEVICE_ID);
540
541     XAdcPs_CfgInitialize(XAdcInstPtr, ConfigPtr,
542                         ConfigPtr->BaseAddress);
543
544     /*

```

```
535         * Self Test the XADC/ADC device
536         */
537         Status = XAdcPs_SelfTest(XAdcInstPtr);
538
539
540         /*
541         * Disable the Channel Sequencer before configuring the Sequence
542         * registers.
543         */
544         XAdcPs_SetSequencerMode(XAdcInstPtr, XADCPS_SEQ_MODE_SAFE);
545
546     }
```