**1. Create data:**

Set up a sine wave with 1V amplitude and frequency = fc using vpulse in ngspice. It is good practice to set the sim up initially for coherant sampling which is done using my "Coherant_Sampler" spreadsheet, checked into the below git location:

https://github.com/SLICESemiconductor/OpenSourceTool_Examples/tree/main/fft_analysis

E.g. For fc ~ 1MHz, where fs = 40MHz, and resolution = 10b => fc = 1.0546875M with the fft window length = 25.6us.

**2. Linearize data:**

ngspice like most simulator engines is event driven. As a result the time steps will be non-linear. To linearize them I use the linearize function in ngspice (see section 17.5.43). This function requires 3 input variables:

- lin-tstart .... the time point to start linearising at
- lin-tstop ... the time point to stop linearising at
- lin-tstep ... the linear time step

Following from the above example where the fft window = 25.6us, I run the sim for 26.6us and set lin-tstart = 1u and lin-tstop = 26.5u. Since fs = 40MHz, lin-tstep = 1/40Meg.

Because this produces 10b resolution the resulting linearised vector should contain only 1024 data points. I say "should" because in reality there appears to be a bug in the linearise funciton which causes it to produce 1 sample more. Therefore, to avoid it producing 1025 data points I set lin-tstop = 26.5u-(1/40Meg).

This bug is present at time of writing but always check for it as if it is fixed you dont need to subtract the final sample obviously. The reason you want exactly 1024 pts is it is to the power of 2 and if your fft solver requires the fft lenght to be to the power of 2 (as the John Green method does), it will zero pad which I dont like (reduces amplitude).

Example code to do this linearising is below:

```
  let lin-tstart = 1u
  let lin-tstop = 26.6u-((1/40Meg)*1)
  let lin-tstep = 1/40Meg
  linearize v(vsquare)
  print v(vsquare) > vsquare_lin_data.txt
```

After running this code, check vsquare_lin_data.txt to verify the number of data points = 1024.

At this point you can choose to perform your fft analysis in ngspice or octave.

**3. Running ffts in octave:**

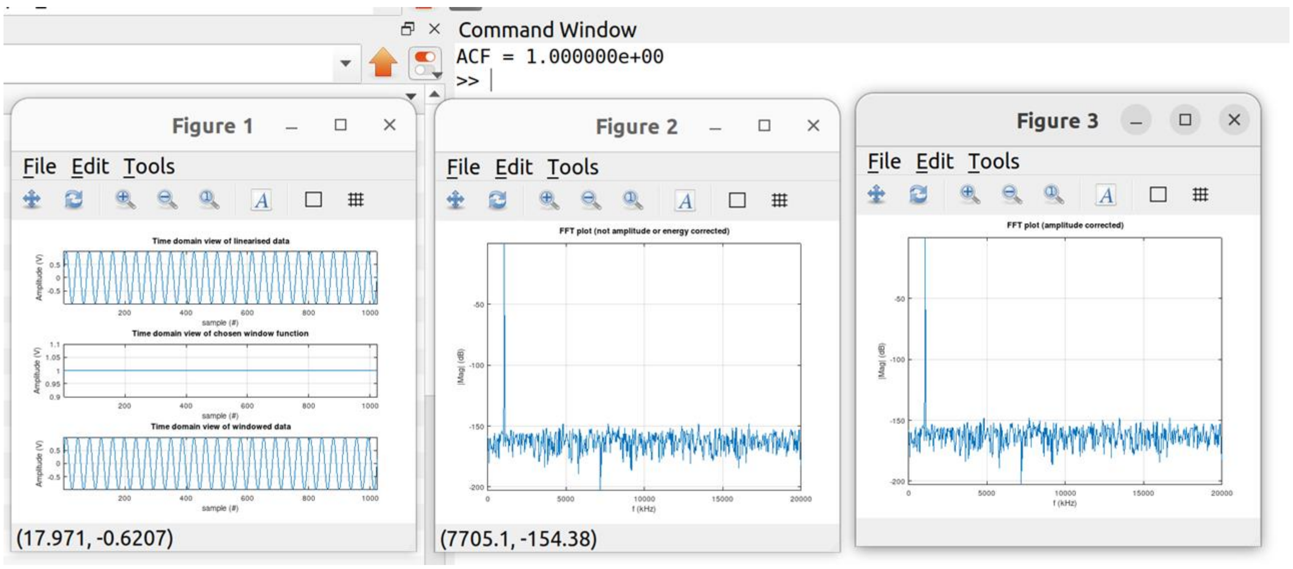
Octave script "fft_analyser.m" to run ffts is checked in at the below git location:

https://github.com/SLICESemiconductor/OpenSourceTool_Examples/tree/main/fft_analysis

This reads “vsquare_lin_data.txt” and performs an fft on it using one of the following windows:

- rectangular
- bartlett
- hann
- blackman
- 4term blackman harris

The below shows the script output using a rectangular window:



```
Command Window
ACF = 1.000000e+00
>> |
```

Figure 1
File Edit Tools
Time domain view of linearised data
Amplitude (V)
sample (#)
Time domain view of chosen window function
Time domain view of windowed data
(17.971, -0.6207)
Figure 2
FFT plot (not amplitude or energy corrected)
|Mag| (dB)
f (kHz)
(7705.1, -154.38)
Figure 3
FFT plot (amplitude corrected)

The rectangular window effectively corresponds to no window which can be used on the original data since it was coherantly sampled (no spectral leakage occurs).

If we sample in-coherantly however, this will not be the case. To do this, set lin-tstop 1 quarter period earlier (to achieve maximum in-coherancy), as per below:

let lin-tstop = 26.6u-((1/40Meg)*1)-0.25u

Re-running the script on vsquare_lin_data.txt clearly shows spectral leakage to occur, as per below:

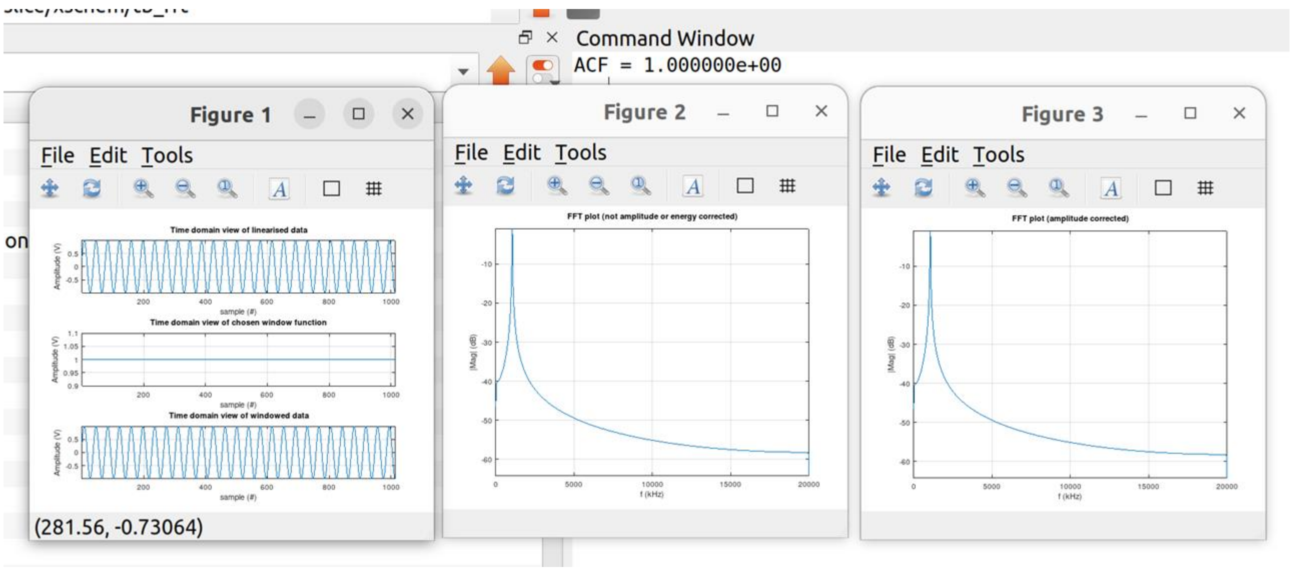


Figure 1
Time domain view of linearised data
Time domain view of chosen window function
Time domain view of windowed data
Amplitude (V)
sample (#)
Figure 2
FFT plot (not amplitude or energy corrected)
|Mag| (dB)
f (kHz)
Figure 3
FFT plot (amplitude corrected)

```
Command Window
ACF = 1.000000e+00
```

(281.56, -0.73064)

To avoid this we use windowing. Below shows the output using the nations favourite 4term blackman harris window:

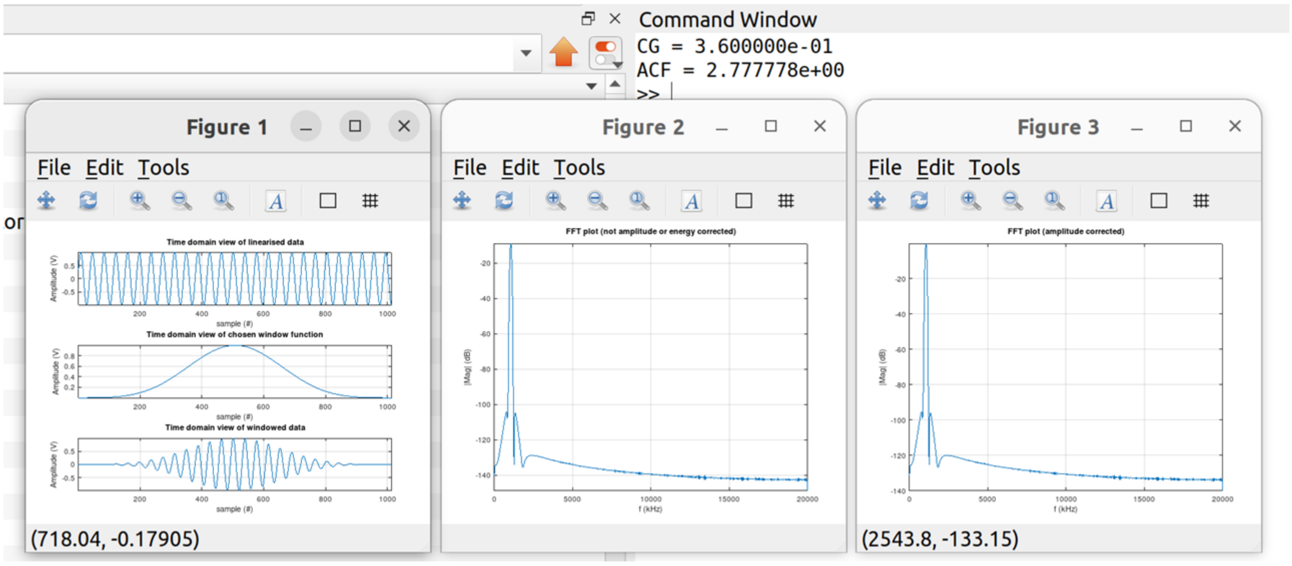


Figure 1
Time domain view of linearised data
Time domain view of chosen window function
Time domain view of windowed data
Amplitude (V)
sample (#)
Figure 2
FFT plot (not amplitude or energy corrected)
|Mag| (dB)
f (kHz)
Figure 3
FFT plot (amplitude corrected)

```
Command Window
CG = 3.600000e-01
ACF = 2.777778e+00
>>
```

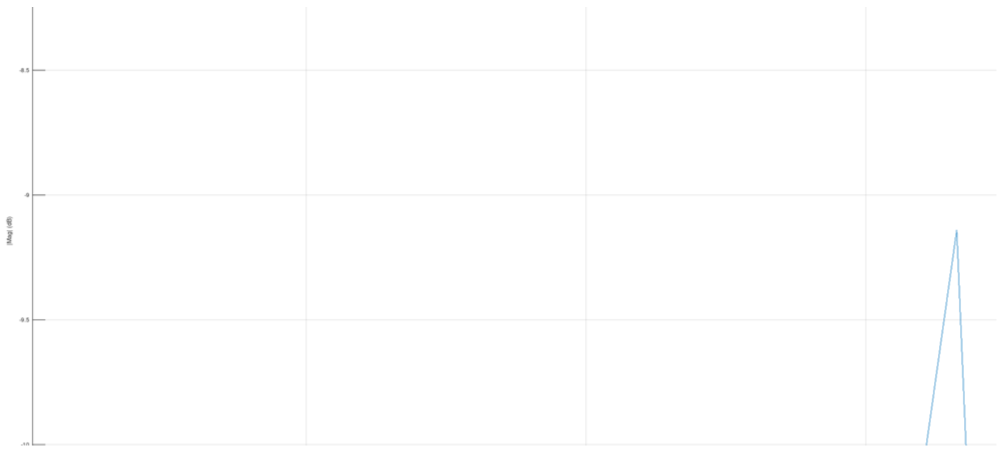
(718.04, -0.17905)

(2543.8, -133.15)

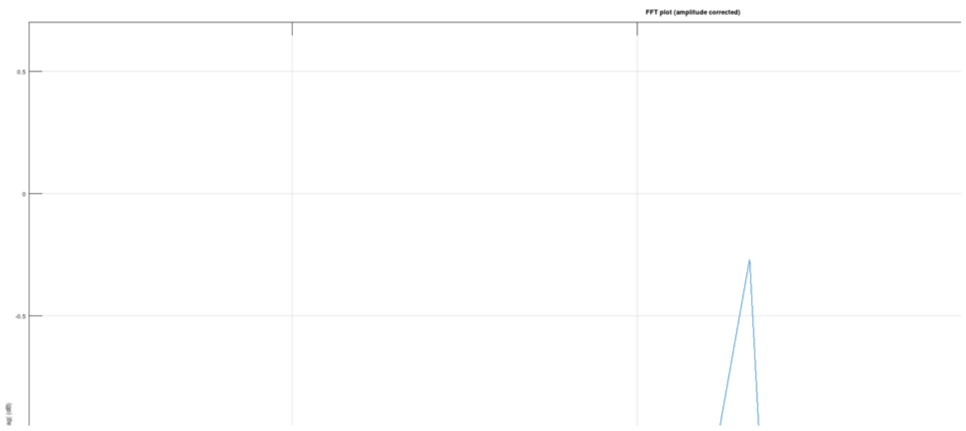
You will notice the "CG" and "ACF" outputs in the command window. These stand for coherant gain and amplitude correction factor. Each window function has a specific coherant gain which is < 1 and so always reduces the signal amplitude (see blackman harris paper). By multiplying the data point in the signal bin only by 1/CG (=ACF) you will restore the correct amplitude as shown below. However you cant just scale up the signal as then you would be falsely increasing your SNR. If you scale up the signal you need to scale up the noise by the same amount to preserve your SNR. As a result, the ACF needs to be applied to every spectral component (as is done in the octave script).

Figure 2 (from the script) outputs the fft data which is not amplitude corrected showing the signal to be ~ 10dB down.

Figure 3 (from the script) outputs the fft data which is amplitude corrected showing the signal to be ~ 0dB as expected for a 1V signal.



At present you have to manually set the CG. In fairness I could probably automate this step in the script but like to leave it manual to force the user (i.e. me!) to be mindful about CG and ACF.



To make using the octave script as seamless as possible, save it in the same directory that you run ngspice in. That way vsquare_lin_data.txt saves in the same directory as the octave script allowing you to have the octave script open side by side where one press of a button and it runs.

Note: There is a lot of theory about windowing which is contained in my notes. These deal with the best possible windows, their tradeoffs and how to perform SNR calculations from the fft data. Not contained here as it is beyond the scope of this document.

## 3. Running ffts in ngspice:

ngspice itself can also run ffts. To do this, first set the window as follows:

```
 set specwindow = none
 *set specwindow = rectangular
 *set specwindow = bartlet
 *set specwindow = blackman
 *set specwindow = blackmanharris
 *set specwindow = hanning
 *set specwindow = gaussian
 *set specwindow = flattop
```

Above shows all the available windows, commenting in only the window I want to use. In this case it is none. Interestingly they include “none” and “rectangular” which are the same window! You’ll also notice they include blackmanharris which corresponds to a 4term blackmanharris and was included as per my request in the below thread:

https://sourceforge.net/p/ngspice/discussion/127605/thread/03b82f8bf2/

Then simply perform the fft with the below command:

```
fft v(vsquare)
```

Where v(vsquare) is taken from step 2 which produced data with a linear time step using: linearize v(vsquare). For post processing reasons (will be shown why later), include the below lines:

```
print v(vsquare) > vsquare_fft_data_black.txt
plot 20*log10(mag(1*v(vsquare)))
remzerovec
write tb_OTA_sp5.raw
```

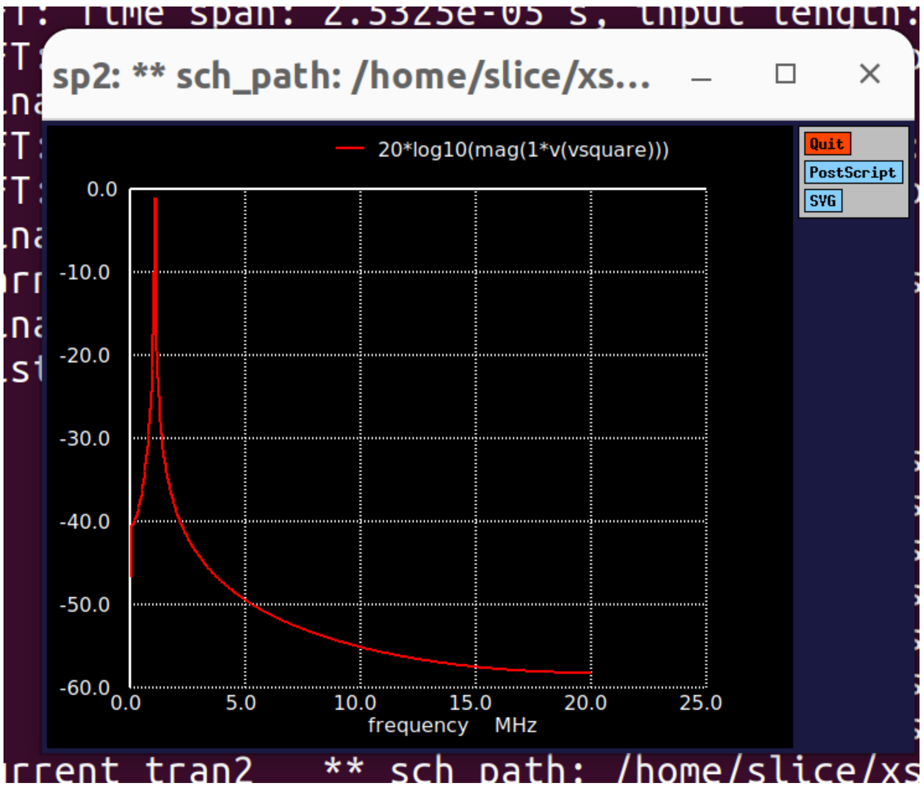
**4. Viewing fft spectral plots:**

FFT spectral plots can be viewed in ngspice, xschem or octave.

4.1 Viewing fft spectral plots in ngspice:

Include the line: plot 20*log10(mag(1*v(vsquare)))

Output will be like (data is incoherantly sampled):



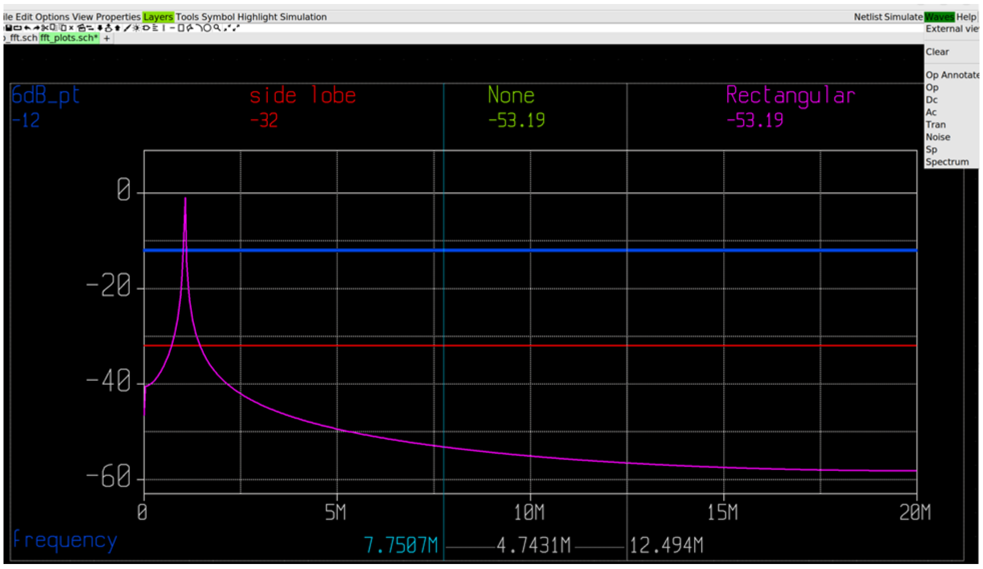
But as with all ngspice plots, there's nothing you can do with this plot!

## 4.2 Viewing fft spectral plots in xschem:

Write the data to a raw file as per below:

```
  remzerovec
  write tb_OTA_sp2.raw
```

Open the waveviewer in xschem and select waves / Sp.
Select that raw file and the spectrum will appear which you can zoom and place horiztontal / vertical cursors on.

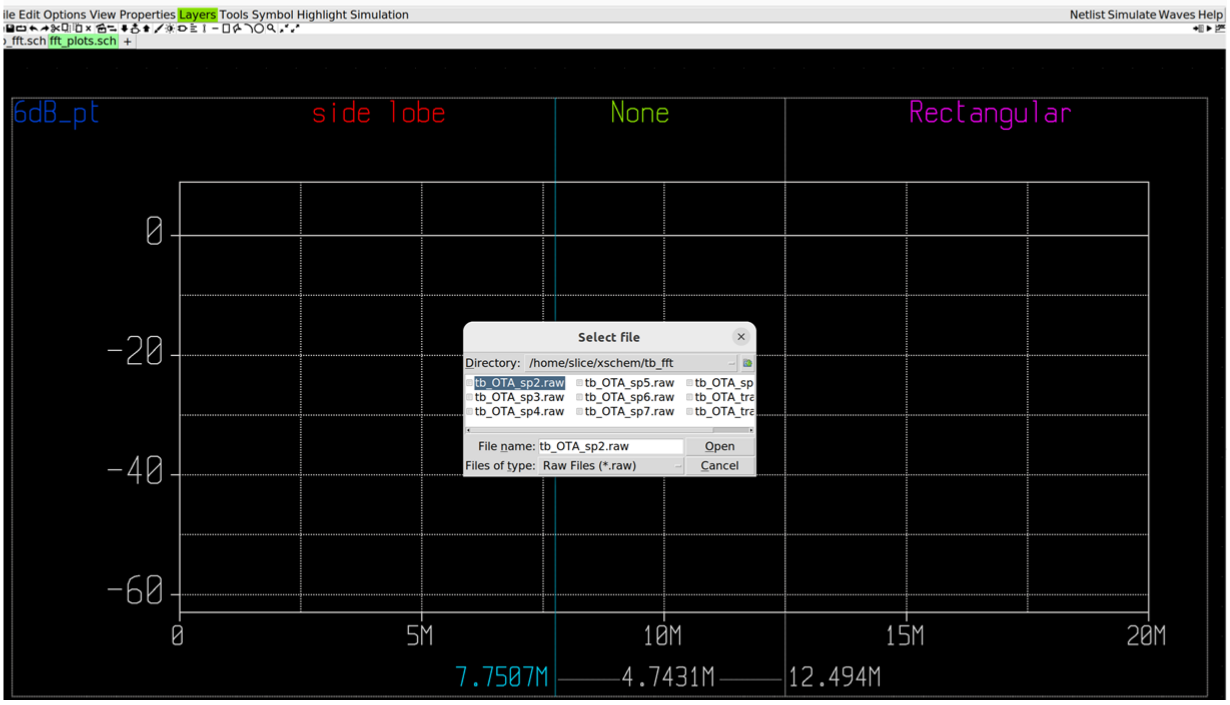


Suppose you run multiple fft windows, writing the output from each one to a new raw file, using the below code:

```
** 3. FFT (none) **

  set specwindow = none
  *set specwindow = rectangular
  *set specwindow = bartlet
  *set specwindow = blackman
  *set specwindow = hanning
  *set specwindow = gaussian
  *set specwindow = flattop

  *spec 0 20Meg 9.765625e3 v(vsquare)
  fft v(vsquare)
  print v(vsquare) > vsquare_fft_data_none.txt
  plot 20*log10(mag(1*v(vsquare)))
  remzerovec
  write tb_OTA_sp2.raw

  ** 4. FFT (rectangular) **
  setplot tran2
  *set specwindow = none
  set specwindow = rectangular
  *set specwindow = bartlet
```

```
*set specwindow = blackman
*set specwindow = hanning
*set specwindow = gaussian
*set specwindow = flattop

*spec 0 20Meg 9.765625e3 v(vsquare)
fft v(vsquare)
print v(vsquare) > vsquare_fft_data_rec.txt
*plot 20*log10(mag(1*v(vsquare)))
remzerovec
write tb_OTA_sp3.raw
```

In this code, we use the window "none" whose raw data is written to "tb_OTA_sp2.raw" and the window "rectangular" whose raw data is written to "tb_OTA_sp3.raw". When we open the waveviewer I select file "tb_OTA_sp2.raw" as shown next.


ile Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
fft.sch fft_plots.sch +
6dB_pt
side lobe
None
Rectangular
0
-20
-40
-60
0
5M
10M
15M
20M
7.7507M
4.7431M
12.494M

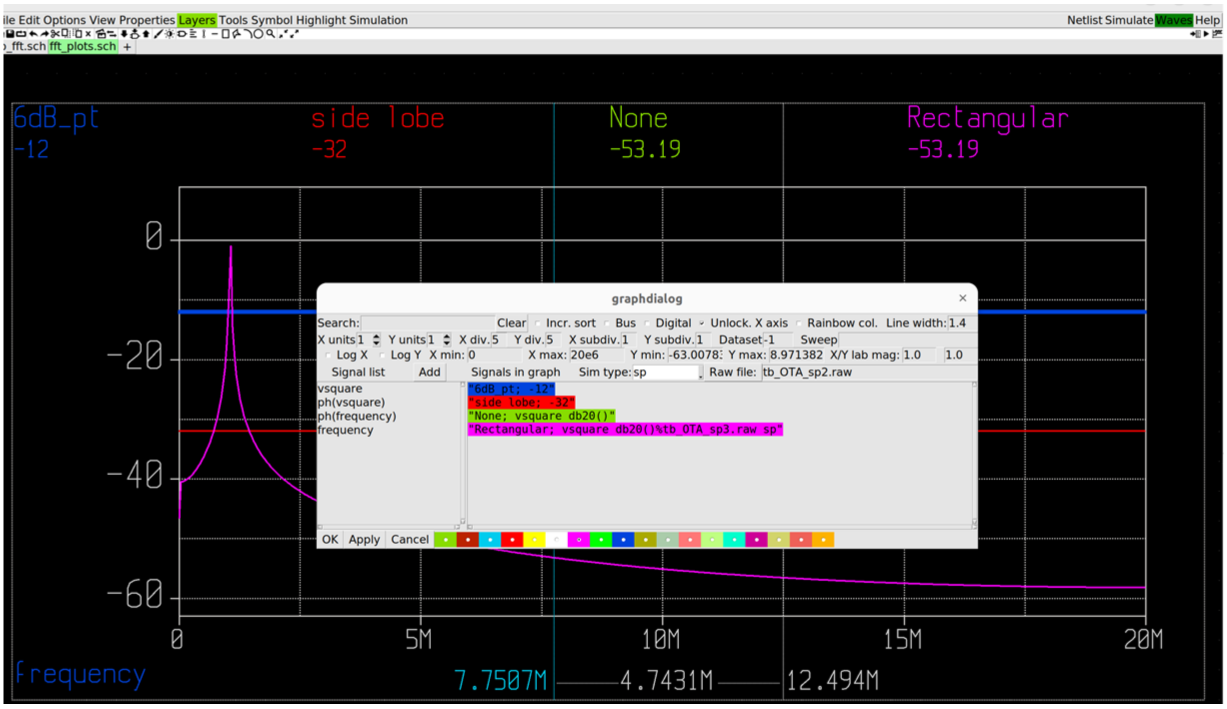
Select file

Directory: /home/slice/xschem/tb_fft

tb_OTA_sp2.raw tb_OTA_sp5.raw tb_OTA_sp
tb_OTA_sp3.raw tb_OTA_sp6.raw tb_OTA_tra
tb_OTA_sp4.raw tb_OTA_sp7.raw tb_OTA_tra

File name: tb_OTA_sp2.raw
Files of type: Raw Files (*.raw)

Then I can superimpose the data from "tb_OTA_sp3.raw" by specifying that name after as shown below:



```
Search:                Clear   Incr. sort   Bus   Digital   ✓ Unlock. X axis   Rainbow col.   Line width: 1.4
X units 1   Y units 1   X div. 5   Y div. 5   X subdiv. 1   Y subdiv. 1   Dataset -1   Sweep
Log X   Log Y   X min: 0   X max: 20e6   Y min: -63.0078   Y max: 8.971382   X/Y lab mag: 1.0   1.0
Signal list   Add   Signals in graph   Sim type: sp   Raw file: tb_OTA_sp2.raw
vsquare             "6dB_pt; -12"
ph(vsquare)         "side lobe; -32"
ph(frequency)       "None; vsquare db20()"
frequency           "Rectangular; vsquare db20()%tb_OTA_sp3.raw sp"
```

Note how the expression is built up, as described at:

https://xschem.sourceforge.io/stefan/xschem_man/graphs.html

Where the expression syntax is:

"alias_name;operand operand operator ..."

So from the above example:

- alias_name = None or Rectangular
- operand = vsquare
- operator = db20()

Now additional to this you can specify the raw file, other than the one you have loaded in, making the more general syntax as:

"alias_name;operand operand operator%<rawfile_name> rawfile_type (e.g. ac/sp)"

This is a good example as it shows rectangular and none to give identical results as one would expect leading to the question of why on earth does ngspice specify to use either window!?!

<u>4.3 Viewing fft spectral plots in octave:</u>

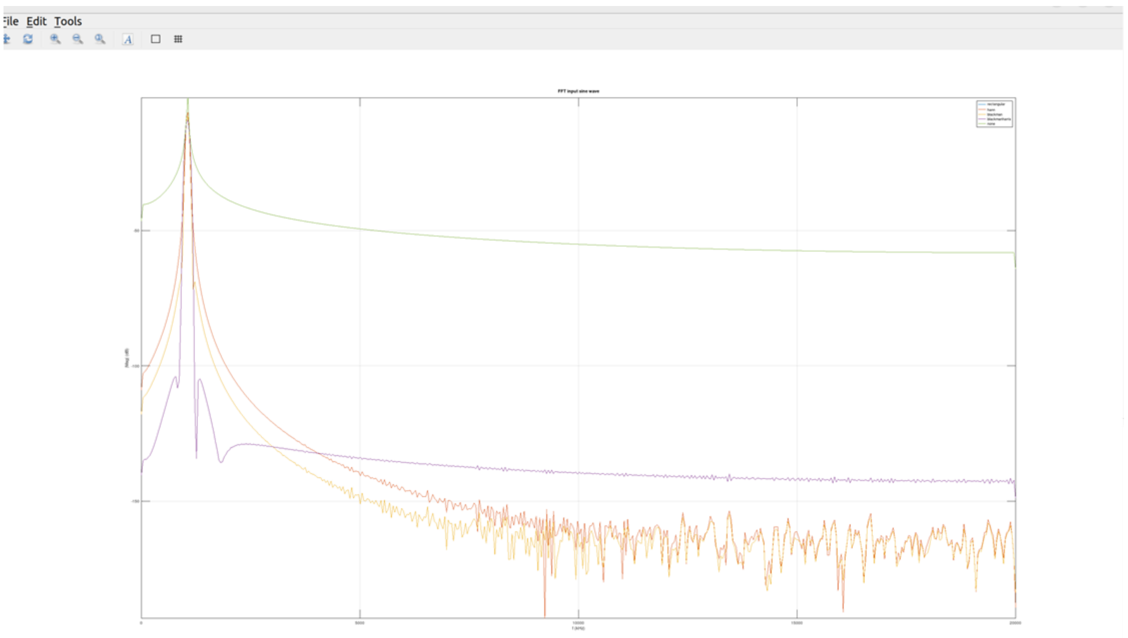
fft plots can be viewed in octave with the file "ngspice_fft_plotter.m" checked in at:

https://github.com/SLICESemiconductor/OpenSourceTool_Examples/tree/main/fft_analysis

As ngspice automatically amplitude corrects (which it shouldnt in my opinion as that sort of thing should be left to the user), there is not ACF included in this script since it is not necessary (and I havent bother to amplitude "uncorrect").

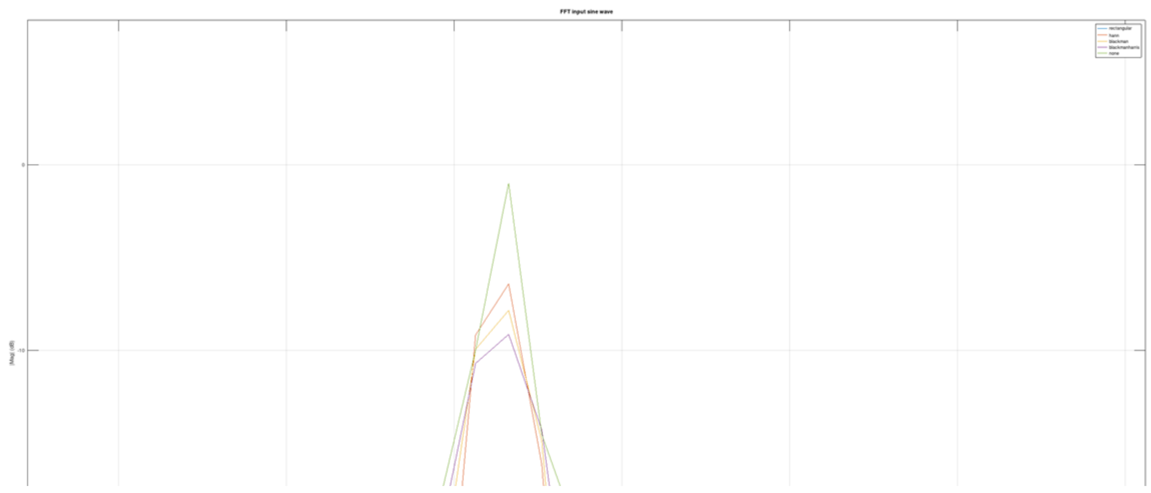
Personally I like to use octave as it allows me to do lots of comparative analysis. For example, the code “fft_windowing_comparisons.m” (checked in at the same location) allows me to perform ffts using multiple windows and superimpose them on the one plot as shown below.

For this



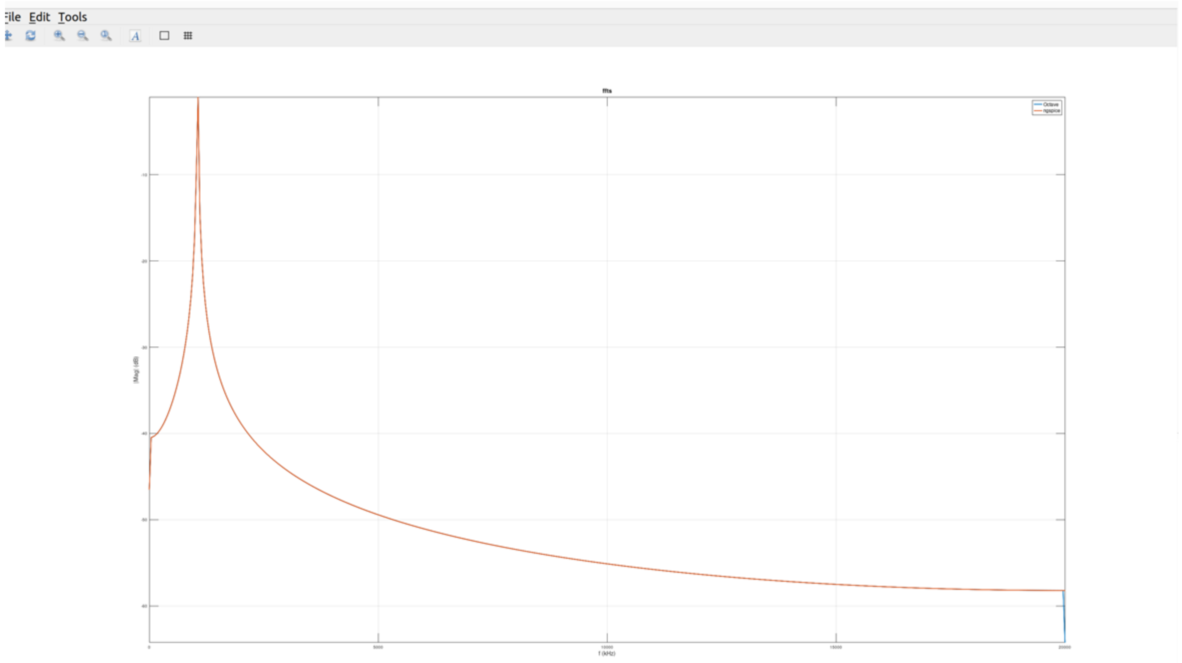
sims I have analysed the following windows: None / Rectangular / Hann / Blackman / Blackmanharris.

This script does not amplitude correct since its function is to provide comparisons and so not amplitude correcting allows us to compare the different coherant gains for each window, as shown below:



Note: You could also do this in xschem as shown in section 4.2.

Then the code “octave_ngspice_fft_comparison.m” can be used to compare the fft outputs from ngspice with those from octave. Below shows this comparison for the rectangular window:



Note the plot has been nicely adjusted using the adjfig script which should be contained in the same folder so that the code can pick it up.

This script was instrumental in evaluating the newly added blackmanharris window as was the main outcome from the below thread:

https://sourceforge.net/p/ngspice/discussion/127605/thread/03b82f8bf2/

**5. Example:**

All the require files for fft analysis are checked in to the below:

https://github.com/SLICESemiconductor/OpenSourceTool_Examples/tree/main/fft_analysis

This performs a 10b fft on a ~ 1MHz (un)coherantly samplied sine wave (fs = 40MHz) using a variety of windows.

**6. Conclusion:**

Honestly, there were just too many bugs with ngspice fft analysis for me to feel 100% comfortable with it. Example bugs were (the linearise function producing 1 extra code, the fft outputted an amplitude corrected waveform by default, not having the blackmanharris window). I get the feel that Dietmar is good but not extremely experienced with ffts. Therefore, I would advise running ffts in octave instead, using the linearise function to create the linear time steps (we can live with the 1 extra sample bug by a simple code correction. Its an old function so wont have any other bugs). There is no additional overhead required provided the octave script is located in the same directory as the linearise steps output file (and adjfig script). You simply run ngspice, then go to octave which will already be open, and run from there.

Some people run ffts in Matlab even when using Cadence as the plots are nicer. In some cases (e.g. Hassan with Snysps), some peoples confidence isnt high enough to use the EDA tool for fft analysis and so use Matlab instead. This is similiar to my situation – I am more confident with Octaves fft analysis (because it is taken from Matlabs) than ngspices fft capability (as it still feels like a WIP).

As such, I run ffts in Octave, using ngspice only for the linearising step, which makes use of an old robust function (linearise). When it comes to plotting then, while you’re in Octave, you may as well run the fft plots there. But you can also do it in xschem if required. Worst of all, would be to use ngspice because of its poor plotting capability.