Oscillator Design Using LTSpice

By David Stockton, GM4ZNX

Modern RF design is heavily reliant on simulation. Universities have shifted emphasis from hardware labs to computer simulations. Simulators work well on amplifiers, filters, modulators, mixers and so on, but not as well on oscillators. This leaves even graduate-level RF engineers rather light on the subject of oscillators.

Such a void is a back-door opportunity to oscillator analysis. Simulation has not been used much by amateurs because it is computationally intensive. Most recent PCs have plenty of power for this task, however. You may not be aware of how powerful your machine is, because things like operating systems and word processor software have become progressively inefficient. Set your computer loose on a math-intensive task like circuit simulation and see what a beast has been hiding behind the flashy graphics and zillions of type fonts.

The simulator needs to be able to handle oscillators, and it needs to be affordable on an amateur budget. High-end RF CAD packages usually contain tools for handling oscillators, but they aren’t easy to use and many are afterthought additions that don’t mesh with the rest of the package. Oscillators are diabolical things to simulate. Even if you can afford to shovel money at the problem, it still takes a bit of creativity to use a full “bells and whistles” simulator on an oscillator and for the results to make sense. (For detailed information on simulation, see the chapter on **Computer-Aided Circuit Design**.)

An oscillator is a simple loop with signal flowing round and round in it. We can view the signal flow as being many components adding together at some point at some time. We can think of these components as being different vintages. This component came from thermal noise X microseconds ago and has been round the loop Y times, and that component… you can see the pattern. This is hard to analyze because all components get summed, not separated. So instead of thinking of our oscillator as a single stage with everything distributed in time, let’s distribute it in space as well.

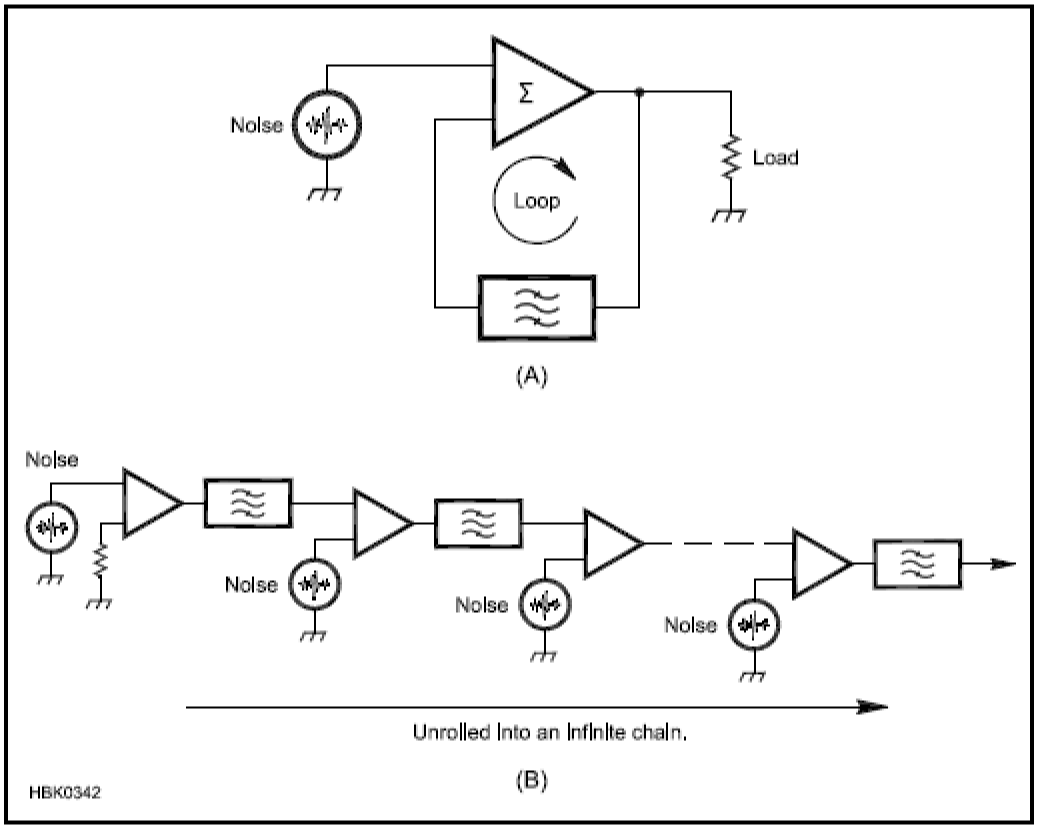

Figure 1 — The “Swiss Roll” visualization used by GM4ZNX. The amplifier loop is “unrolled” to permit its signals to be observed.

The GM4ZNX “Swiss-Roll” technique is to think of a rolled-up jelly cake and unroll the oscillator circuit as illustrated in **Figure 1**. We break the loop open, which leaves us with a tuned amplifier circuit with an input and an output terminal. In a closed oscillator loop, the output terminal would drive the input terminal. In the unrolled circuit, the output terminal drives the input of a second stage and so on.

Instead of infinitely looping signals in a single stage, we now have a single signal flow down an infinite string of stages. This doesn’t sound any better, but we only need to handle a couple of hundred stages. Think about that… we are going to simulate a 200-stage tuned amplifier! But all the stages are identical, so we only have to draw one stage and have the others built as hierarchical repeats. On a 2005 model laptop, a simulation takes several minutes, which is fine.

Affordable RF analysis programs are normally limited to linear simulations and can’t handle the oscillator’s amplitude control mechanism. We also need a simulator that can handle random noise at the same time so the choice of tools becomes very limited. One example of a suitable simulation package, based on *SPICE*, is available free from the Linear Technologies website, **www.linear.com**. (The program is called *SWCADIII* or *LTSpice*.) It is optimized for the analysis of switch-mode power supply (SMPS) circuits. This is very good for our purposes. In an SMPS things ring at high frequencies, and slow things settle. *LTSpice* has some trickery which allows it to easily handle RF speed events, while running a simulation efficiently over a long enough period for control loops to settle. This is what makes it efficient for oscillators. Our oscillator circuits are run at such large signal levels that as the oscillator reaches its long term output, it will affect bias conditions, which are slow-damped by decoupling components. (Again, refer to the **Computer-Aided Circuit Design** chapter for more information on circuit simulation.)

LTSpice can handle noise, and it can also handle hierarchical schematics. We draw our basic oscillator circuit once, and create a symbol for it. We can then use that symbol in a higher level schematic. For our worked example of an oscillator analysis, a single-stage schematic and symbol were created, then a higher level, a 10-pack of those symbols was created. The top level was made of 20 10-packs. Files for these designs are in the *Handbook's* companion CD, but to be able to set them up in *LTSpice* you will need to read *LTSpice's* help files on hierarchical designs. Hierarchical schematics save the need to draw our oscillator stage 200 times. It also allows editing a parameter on one schematic to change all 200 stages at once. Without this feature, things would be tedious.

Figure 2 shows a basic oscillator circuit. The transistor is a bipolar type, operated in common-base mode. The resonator is an LC tank with a capacitive tap. The common-base amplifier gives less than unity current gain, low input impedance and high output impedance. There has to be an impedance transformation to make the loop have enough gain to oscillate, and the capacitive tap is it. Instead of having bias resistor networks, the stage has a voltage source to set the collector bias voltage, and a current source to set the quiescent emitter current. You can just type in your bias values without having to calculate resistor values. One drawback of *LTSpice* is that it has few RF parts in its libraries, but a search on the Internet will find user groups and guidance on editing models for other *SPICE* variants to work with *LTSpice*. The 2N4142 in the supplied library is used in this example. A 5 V collector bias and 0.2 mA emitter bias have been chosen as starting points.

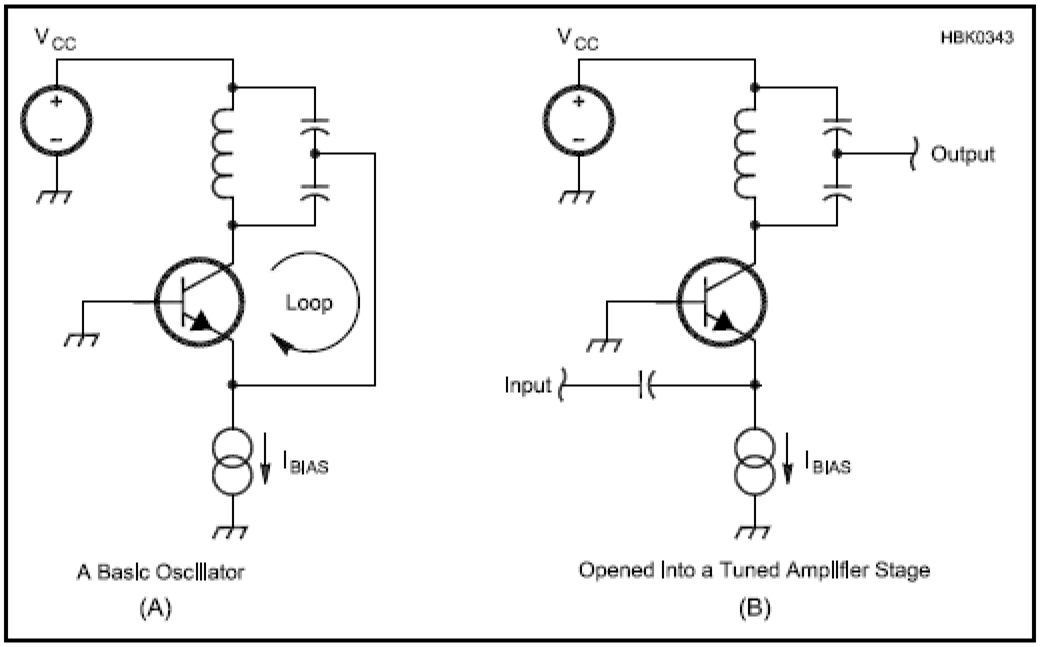

Figure 2 — The basic oscillator circuit (A) is built on a tuned amplifier circuit (B).

In designing this circuit, we must create a stage gain greater than unity, and allow the operating level to rise until that gain is compressed to exactly unity. With a bipolar transistor at RF, we do not want it to go into saturation. Instead, we want it to go into cut-off before that can happen. Set the emitter current for the level you want, and then make sure there is enough collector bias voltage that the instantaneous collector voltage is always a few volts more positive than the emitter. The L/C ratio of the tank, and the capacitive tap ratio are surprisingly tolerant, the circuit will oscillate with these parameters varied over an appreciable range. If your goal is low drift, then you should go for high capacitance in the tank, to dilute the effect of strays. If your goal is low phase noise, then you should try to create as high a tank Q as possible.

Because oscillators of this type use nonlinear device operation to control amplitude, do not try to get low levels of harmonics on their output — the harmonic creation mechanism is fundamental to the oscillator. Try an oscillator with a detector and feedback loop to control its level instead. The square-law characteristic of JFETs is an excellent level-stabilizing mechanism and accounts for their popularity in oscillators. However, their wide range of I_{DSS} for any given type, can be troublesome. MMIC amplifiers with internal feedback are probably the worst choice of device, but some types can be used in special circumstances.

In the simulation circuit provided here you will find some odd things. The random number source is of uniform density, so several of them are added together to approximate Gaussian noise. The collector load resistor which represents the output of the oscillator is split into two parts, a resistor to ground on the output is needed to define the dc potential of a node between the capacitors of adjacent stages (or else *SPICE* complains) and it also loads the tank via the tap.

When you have got all the schematics in the right directories for *LTSpice* to use them, and run the simulation, you will have an immense database of all waveforms at all nodes. You can now probe around the top level schematic. The first transistor in the chain acts as an emitter-follower so you can see its noise voltage coming out of its input if you probe there. The way the random number generators work is tied to the simulation time clock and you will see the first random number generator turn on at about 2 μs into the simulation, followed progressively by the others. This is just an oddity of the realization, but it does illustrate the summation of differently-seeded generators to approximate Gaussian distribution of noise.

Probing along, 10 stages at a time, you will see bigger and bigger noise waveforms until you start to notice the effect of the cumulative selectivity, as it starts to look like a noisy sine wave. Further along you will see the level begin to stabilize. If you zoom out to a larger time-scale, you will see that the sine wave seems to have low frequency random noise AM applied to it. As you probe further along you will see this “AM” compressed until it isn’t seen, by the level control action of the oscillator and progressively narrowing bandwidth. Long before you get to the end of the 200 stages, you will have the appearance of a stable sine wave. Remember that the taps earlier in the structure represent the signal in a closed loop oscillator at times close to when it started-up. In a running oscillator, compression eventually reduces the gain of all amplification passes of all signal components. Fourier transform conversion can be used to convert these time-pictures into spectrum displays, if you want to explore further.

Among the accompanying files there is also a simulator that takes a single stage and plots its compression characteristic. It could be improved by adding earlier and later stages to make the source and load impedances more representative. The example oscillator can also have a model of a quartz crystal added in series with its output, to simulate a crystal oscillator. Explore! And have fun.

How can a lower phase noise oscillator be designed? Firstly if you use a lower noise transistor, it will take more stages of gain and filtering to bring it up to the stable output level.

The greater repetition of filtering before noise reaches a significant level narrows the bandwidth of the output signal. Secondly, if you can engineer a higher-Q tank circuit, each pass through the tank has a greater narrowing effect.

The simulation files for this section are included with this article:

SwissRollDemoBasicStage.asc — (The oscillator stage. Editing this changes every stage)

SwissRollDemoBasicStage.asy — (The symbol file for use in higher level schematics)

SwissRollDemo10Pack.asc — (10 basic stages as a building block)

SwissRollDemo10Pack.ass — (The symbol for a 10 pack)

SwissRollDemoTopLevel.asc — (This is the real simulation, uses the above as components)

SwissRollDemoCompression.asc — (Crude simulation showing stage gain compression)

SwissRollDemoStageBandwidth.asc — (Crude simulation showing filter Q of one stage)