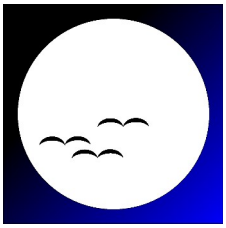

QSpice C-Block Components

Shared Resources & Reference Counting

Document Revisions:

2024.01.11 – Initial Version.

Caveats & Cautions

First and foremost: Do not assume that anything I say is true until you test it yourself. In particular, I have no insider information about QSpice. I've not seen the source code, don't pretend to understand all of the intricacies of the simulator, or, well, anything. Trust me at your own risk.

If I'm wrong, please let me know. If you think this is valuable information, let me know. (As Dave Plummer, ex-Microsoft developer and YouTube celebrity (Dave's Garage) says, "I'm just in this for the likes and subs.")

Note: This is the second document in a clearly unplanned series. The prior C-Block Basics document covered how the QSpice DLL interface works. If you don't have a good understanding of the QSpice functions, passed and returned parameters, and the instance data structure, you probably should start there. See my GitHub repository here for the prior document and code samples for this one.

Overview

If you're writing C-Block DLL components that will be used by others, you should expect that users may put multiple instances on a schematic. You should also handle multiple simulation steps. As long as you don't use any global variables, your DLL will likely behave properly without any of the techniques described below.

On the other hand, you may need/want to ***share resources*** between instances and/or across simulation steps. In that case, you'll be forced to use global variables. This document should help you do that reliably.

Note: Sample code is provided in both C and C++ versions. This document doesn't go through the code in detail – hopefully, the comments are sufficient to understand what it does. Instead, this document describes why the code needs to do what it does.

Shared Resources

So, what is a "shared resource?"

We might define shared resources as anything that:

- Is opened, allocated, and/or initialized once at the beginning of a simulation.
- Potentially used by multiple component instances during a simulation run.
- Must be closed, de-allocated, and/or finalized once at the end of the simulation.

As a concrete example, consider logging data to a file. We'll need to:

- Open the log file when the simulation begins.
- Write data to the file from one or more instances during one or more simulation steps.
- Close the log file when the simulation ends.

Sounds simple enough. As it turns out, QSpice doesn't make it that simple.

The First Problem

When QSpice calls our DLL ***evaluation function***, we test `*opaque` and, if it's a null pointer, we allocate per-instance data and return that address. So, we definitely know that we're creating a component instance. What we *don't know* is whether other component instances have been previously created. So we can't just open the log file – a prior instance may have already done that.

When QSpice calls the `Destroy()` method, we release our per-instance data memory. What we *don't know* is whether other instances are still active. So, we can't just close the log file – we must somehow determine that this is the last `Destroy()` call in the simulation before closing the file.

The First Solution

To enable our DLL to determine how many instances are "active," we can use a global variable as a ***reference counter***. The reference counter is initialized to zero when the DLL is loaded.

When the evaluation function is called and we allocate the per-instance data structure, we test the reference counter. If it's zero, we open the log file. If not, the file is already open. Either way, we increment the reference counter.

When the `Destroy()` function is called, we release the per-instance memory and decrement the reference counter. If – and only if – the counter is now zero, we can safely close the log file.

So far, so good.

The Second Problem

Although I didn't mention it above, we opened the log file with create/overwrite attributes. That is, if the file doesn't exist, it is created; if it does exist, the contents are erased.

When executing a multi-step simulation, each step will delete the file contents. Only the data from the last step will remain in the log file.

QSpice doesn't tell our DLL how many simulation steps will be run or what step we're currently executing. That means that we can't simply open the log file at the start of the simulation and close it at the end. We're stuck with opening/closing the file in each simulation step.

The Second Solution

To solve this, we need to open the file for “create/overwrite” only when the first step is executed and open the file for “append” on all subsequent steps.

We need another global variable, a ***step counter*** initialized to zero. When an instance increments the reference counter from 0 to 1, we increment the step counter. If the step counter was zero, we open the file for “create/overwrite;” otherwise, we open the file for “append.”

OK, that should work and it also provides a step counter to include in the logged data. Cool.

We’re done. You can do a happy dance now.

Other Notes

While working on this, I had a couple of e-mail exchanges with Mike Engelhardt (the QSpice author). I learned that:

- When there are multiple schematic instances of a component, the order of creation within the DLL is arbitrary. That is, if you have two components, X1 & X2, on the schematic, the DLL evaluation function may be called for X1 first or X2 first. From our standpoint, we can’t expect a specific creation order based on anything about the schematic or netlist. (The sample code includes an instance attribute (InstID) to demonstrate this. In my tests, X2/’B’, is instantiated first.)
- Whatever order of component instantiation QSpice selects, calls to QSpice interface functions (the evaluation function, MaxExtTimeStep(), Trunc(), and Destroy()) always occur in the same instance order within that simulation run.

I don’t know how useful the above information may be but, as far as I know, it’s undocumented stuff. I learned something and now you know, too.

I’m stopping here. Let me know where I’ve gone far afield.

--robert