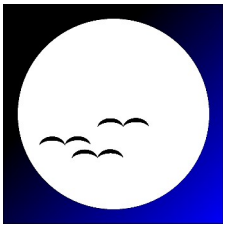

QSpice C-Block Components

DllMain() & Simulation Start/End Tasks

Document Revisions:

2024.01.13 – 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 third document in a clearly unplanned series:

- ***The first C-Block Basics document covered how the QSpice DLL interface works.***
- ***The second C-Block Basics document demonstrated techniques to manage multiple schematic component instances, multi-step simulations, and shared resources.***

This document builds on the prior information so consider them "required reading." See my GitHub repository here for the prior documents and code samples for this one.

Overview

In part two of this rather random series, we covered methods to reliably manage multiple schematic components, multiple simulation steps, and a shared resource (a log file). There was much reference counting and step tracking. There was much opening and closing of the log file. The techniques were convoluted but they were reliable.

Now I have a confession: I lied.

I stated that a C-Block DLL component doesn't have a way to recognize when a simulation starts and ends. I asserted that we couldn't simply open the log file at the start of the simulation and close it at the end. That wasn't true.

We can, in fact, open a shared resource before any calls to the DLL component evaluation function and close that resource after QSpice calls `Destroy()` for the last instance. To correct the record, we'll need to delve into the dark corners of `DllMain()`.

Relevant Microsoft documentation: DllMain entry point and Dynamic-Link Library Best Practices.

Using DllMain()

`DllMain()` is called when DLLs are loaded and unloaded. Since QSpice loads/unloads C-Block component DLLs at the start/end of simulation runs[1], we can use this function to allocate/de-allocate and/or initialize/finalize shared resources[2].

Let's modify the prior C-Block Basics code (`CBlockBasics2_C.cpp`) to open/close the log file in `DllMain()`. The new code is supplied in `CBlockBasics3_C.cpp` and looks like this:

```
FILE *LogFile;       // log file ("shared resource")
const char *LogFname = "@logfile.txt";

...

int __stdcall DllMain(void *module, unsigned int reason, void *reserved) {
  switch (reason) {
  case DLL_PROCESS_ATTACH:   // Initialize once for each new process.
    ::LogFile = fopen(::LogFname, "w");

    /* error handling must be included lest failure locks up component */
    if (!::LogFile) {
      display("Unable to create/open logfile (\"%s\").\n", ::LogFname);
      display("Aborting simulation run.\n");
      return 0;
    }

    display("Logfile opened for overwrite (\"%s\").\n", ::LogFname);
    fprintf(::LogFile, "Toolset=%s\n", "DMC");
    break;

  case DLL_THREAD_ATTACH:   // Do thread-specific initialization.
    break;

  case DLL_THREAD_DETACH:   // Do thread-specific cleanup.
    break;

  case DLL_PROCESS_DETACH:   // Perform any necessary cleanup.
    if (reserved != nullptr) {
      fclose(::LogFile);
      ::LogFile = nullptr;
      display("Logfile closed.\n");
      break;
    }
    break;
  }

  return 1;
}
```

It's all pretty straight-forward:

- We create/open the file when the DLL is loaded and close it when the DLL is unloaded.
- If `fopen()` fails, we display a message in the QSpice Output window and return `false` from `DllMain()` to abort the simulation. (Yes, `stdout` is available from within `DllMain()`.)

  Note: If we wanted to allow the simulation to continue, we would need to ensure that the remainder of the program never tries to write to the invalid `LogFile` handle.

---

1 Multi-step simulations load/unload the DLL only once for QSpice releases after 12-31-2023. Prior versions loaded/unloaded the DLL between each simulation step.

2 There are limitations on what can be done in DllMain(). See Dynamic-Link Library Best Practices for a good summary.

- When the DLL is unloaded, we display a message in the QSpice Output window and close the log file.

The file opens/closes elsewhere in the code were removed since they're no longer needed.

Considerations

Is version 3 (`CBlockBasics3_C.cpp`) better than version 2 (`CBlockBasics2_C.cpp`)?

Maybe. Maybe not.

- V3 is certainly cleaner and marginally faster without the repeated file opens/closes, at least for multi-step simulations. If we had a different type of shared resource to manage – one that had significant allocation/de-allocation or initialization/finalization implications – V3 might be better.
- The V3 code relies on QSpice making a single load/unload of the DLL and no separate thread handling. While I think it unlikely, this is technically undocumented and QSpice could change.
- `DllMain()` doesn't have access to QSpice-passed parameters. The sample code for both versions has hard-coded log filenames. V2 could be modified to use a filename passed from a schematic component attribute. V3 is stuck with a hard-coded filename.

If we think outside the box, there's yet one more approach that fixes that last item.

A Hybrid Approach

What if we want to use a schematic component attribute to pass the log filename to the DLL but also want to open and close the file exactly once?

We could do this to the V2 code:

- Copy the V3 `DllMain()` and modify it to *only* to close the file.
- Modify the V2 evaluation function code to:
  - Use the passed filename attribute to...
  - Open the file...
  - Only when initializing the first instance for the first simulation step.
- Remove the `fclose()` calls from `Destroy()` (as in V3).

With these changes, we get the best of both worlds.

Conclusion

If you made it this far, well, congratulations and thanks for sharing the ride. I hope that you found the information useful.

Do let me know if you find problems or my explanations need improvement.

--robert