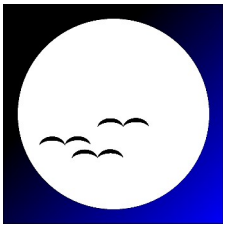

QSpice C-Block Components

Recent Changes (July/August 2024)

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 seventh 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.***
- ***The third C-Block Basics document built on the second to demonstrate a technique to execute component code at the beginning and end of simulations.***
- ***The fourth C-Block Basics document revisited the Trunc() function and provided a basic "canonical" use example.***
- ***The fifth C-Block Basics document revisited the MaxExtStepSize() function with a focus on generating reliable clock signals from within the component.***
- ***The sixth C-Block Basics document examined connecting and using QSpice bus wires with components.***

This document describes recent QSpice changes relevant to C-Block coding. It builds on concepts and definitions in the prior documents so I strongly suggest that you review those first. See my GitHub repository here for the prior documents.

Overview

July and August 2024 releases of QSpice included a couple of nice improvements. I'll describe them briefly below. Subsequent sections provide more detail about subtle differences as well as minimal instructions for updating existing code.

Note: *At this point, past C-Block Basics papers have not been revised to reflect the above changes. Until I get around to that, please keep the below in mind.*

The Display() function (July 2024)

The new `Display()` function is a drop-in replacement for the old `display()` function. It does several things:

- Resolves text corruption issues when displaying text in the QSpice Output window.
- Substantially improves simulation times when outputting text messages.
- Prefixes each output line with the schematic component instance name.
- Suppresses output when the `Display()` call occurs (directly or indirectly) from within a `Trunc()` function call.

The simulation speed improvement should be noticeable for longer simulation runs. However, running simulations from the command line will remain significantly faster than the GUI for larger amounts of text.

The Trunc() Function (August 2024)

The `Trunc()` function has been simplified. It no longer needs to save `uData` output port values and restore them after calling the evaluation function. The code is now less confusing and, in theory, simulations should execute just a wee bit faster.[1]

If you're starting a new component project from scratch using the QSpice code template generator or my QCodeGen tool (using an appropriate template), the above may be all that you need to know.

On the other hand, if you're interested in the subtle edge-cases or want to update existing code, read on.

Display() Function

Output Suppression

As mentioned above, the new `Display()` function suppresses output when QSpice calls `Trunc()`. Since `Trunc()` commonly calls the evaluation function, `Display()` calls in the evaluation function are also suppressed until `Trunc()` returns.

But what if you want to display messages from within a `Trunc()` call when debugging some complicated bit of code? You have a few choices:

- Add the old `display()` code to your project. It's still supported.
- In fact, anything that writes to `stdout` (e.g., `printf()`) still gets pushed to the Output window. You may want to add delays (like the old `display()` does) to reduce possible corruption.

1 "Wee bit faster" would be measured in sub-microseconds.

- Write to a debugging log file (e.g., with `fprintf()`). For serious debugging and large quantities of text, this is the best and fastest option during development.[2]

String Formatting

There is a more subtle difference – one so obscure and unlikely to be an issue that I'm almost embarrassed to bring it up. But I'm nothing if not obsessively thorough….

Technically, when you compile a component with the old `display()` function, the formatting library provided by your component compiler is linked into your component. The new `Display()` function calls the formatting library of the tool used to compile QSpice. If there are differences between the two compiler formatting libraries, you may have issues.

If you run into problems, you can simply use something like `sprintf()` to use your compiler's library to format the string into a buffer. Then call `Display()` with your formatted string as the only parameter.

Updating An Existing Component

Updating an existing component is straight-forward:

1. Delete the old `display()` function code.
2. Delete the following include statements (if not used elsewhere in your code):

```
#include <stdarg.h>
#include <stdio.h>
#include <time.h>
```

3. Add the new `Display()` function declaration after any include statements:

```
extern "C" __declspec(dllexport) int (*Display)(const char *format, ...) = 0;
```

4. Replace the old `display()` calls with `Display()`. That is, simply find/replace "`display(`" with "`Display(`".

That should be it.

Trunc() Simplifications

As mentioned, `Trunc()` no longer needs to save/restore `uData` output elements. But there are subtle differences.

Under The Hood

In the "before days," QSpice internally allocated a single `uData` array for each schematic component instance. This memory location is passed as the `uData*` data parameter to the evaluation and `Trunc()` functions. In `Trunc()` calls, we needed to save/restore the `uData` output port values to ensure that temporary changes didn't later get passed into the evaluation function.

---

2 I've covered using log files in other papers on my QSpice GitHub repository.

QSpice now allocates two `uData` arrays for each component instance. The first `uData` array is the "live" simulation data and passed when QSpice calls the evaluation function directly. The second `uData` array is used for `Trunc()` calls. Since the two arrays are separate, we no longer need to save/restore output port values in `Trunc()`.

Now, code is less confusing. We can't screw up as easily. Simulations should run faster. Excellent.

However, there is one possible complication: The `uData` address passed into the evaluation function is now different when QSpice calls it directly vs when `Trunc()` calls the evaluation function. If your code saves a pointer to `uData` in per-instance data (or, heaven forbid, a global variable), your program will break.[3]

Bottom line: Don't save a pointer or reference to `uData` elements unless you've really thought it through. (You can save/restore pointers/references inside the `Trunc()` call if you really need to do that.)

Updating An Existing Component

Updating existing code is pretty simple. Consider the code in Figure 1.[4]

```
extern "C" __declspec(dllexport) void Trunc(InstData *inst, double t,
    uData *data, double *timestep) {
  const double ttol = 1e-9;

  if (*timestep > ttol) {
    // Save a copy of the output vector
    double      &Vout  = data[4].d;
    const double _Vout = Vout;

    // create temporary copy of instance data
    InstData  tmp  = *inst;
    InstData *pTmp = &tmp;

    // call the evaluation function with the temporary instance data
    cblockbasics4(&pTmp, t, data);

    // implement a meaningful way to detect if the state has changed...
    if (Vout != _Vout) { *timestep = ttol; }

    // Restore output vector
    Vout = _Vout;
  }
}
```

Figure 1: "Old" Trunc() Code

3 You might think that no sane person would write such code. For what it's worth, I did. I had good reasons. But I don't claim to be sane….

4 Adapted from C-Block Basics #4 ("Revisiting Trunc()").

The highlighted code saves and restores the `uData` output port data. This is no longer required. Delete your code's corresponding save/restore bits, recompile, and test.

That should do it.

Conclusion

I hope that you find the information useful. Please let me know if you find problems or suggestions for improving the code or this documentation.

--robert