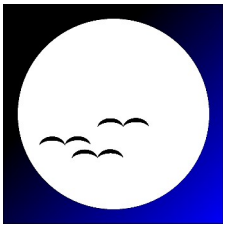

QSpice C-Block Code Templates

In Search of More Modern Code

Document Revision: 2023.11.14

The Background

The QSpice C-Block code generator is a neat feature. However, I'm not a fan of the C-nature of the code it produces. I'd like to use C++ features to reduce code clutter and reflect more modern techniques.

So, each time that I start a new C-Block project, I end up revising the generated code template. It doesn't take long but I'm impatient. Time is precious. Life is short. Minutes count.

The Plan

Here's my plan: I'll create a custom C-Block template that has the changes that I typically make. For code that I plan to share, I usually try to make it compatible with DMC (the installed QSpice compiler). But, if I want to use modern C++ features, I use Microsoft Visual C/C++ or MinGW. Maybe I'll need two or three versions of the template. Or a single template with some pre-compiler magic to handle all three compilers. We'll see. (I did. See The Compiler Support below.)

The Template Changes

Here are the changes to the QSpice-generated template that I plan to make:

- Revise the `#include` C libraries to use C++ versions of the libraries.
- Replace the C `malloc()/free()` per-instance allocations with C++ `new/delete`. Remove the now-unneeded `bzero()` function.
- Add versioning information to the `display()` function and rename it to `msg()`. (Four fewer characters is four fewer opportunities to mistype. I mistype a lot.)
- Change the per-instance structure to a standard name (`InstData`). (QSpice doesn't actually know or care about the structure name.)
- `#define UDATA()` for convenience. (See The UDATA() Thing below). Maybe move the `uData union` declaration/definition to a separate header file to reduce clutter in the *.cpp module.
- Modify the template function signatures to remove unnecessary `struct` and `union` keywords. (C++ treats `struct` and `union` names as `typedefs`.)

- Maybe modify the template function signatures to use references and save typing those clumsy pointer de-references. (Did I mention that I mistype a lot?)
- Add per-instance memory allocation error detection and message to the evaluation function.
- Add my DbgLog tool to the configuration.

The Usage

To use the template, first create a C-Block hierarchical block and generate a template using QSpice. Then:

- Edit the name of the evaluation function in Cblock.cpp to match the generated function name.
- Copy the `uData` array offsets from the evaluation function into the `UDATA` definition and modify as needed. (I'll get to this, I promise.)
- Rename Cblock.cpp to the generated *.cpp file name.

Thereafter, if you add, delete, or modify the C-Block ports or attributes:

- Regenerate a temporary C-Block template to get the new `uData` array offsets.
- Copy the changes to the `UDATA` definition and modify as needed.

Random Tip: You can temporarily rename the target DLL in the hierarchical block before regenerating the C-Block template code. Copy the `uData` stuff into the "real" *.cpp and restore the name in the hierarchical block.

The Cblock.cpp template contains all of the optional QSpice functions and code. When the project is debugged and finalized, you can remove any unneeded template functions (`MaxExtStepSize()` and/or `Trunc()`) and the DbgLog tool.

Random Note: QSpice does not require the `MaxExtStepSize()` and/or `Trunc()` functions to be present. That is, when a simulation is started and QSpice loads the DLL, it checks to see if the DLL contains the functions. If not, it doesn't try to call them. During development, there's no harm in leaving them in the code but do delete them if not used in production.

The UDATA() Thing

A pointer to a `uData` array is passed into the evaluation and `Trunc()` functions. The QSpice template generator puts code into these functions to access copies or references to the array elements. That's two functions that use the array and I want to make sure that I don't fail to change both when changing the offsets/types. If I create additional functions that need `uData`, that's more places that I won't mess up.

To ensure a "single definition" approach, I use the pre-compiler to define the `uData` array access stuff. For example, given this code from the template:

```
double  In            = data[0].d; // input
int     SomeAttribute = data[1].i; // input parameter
double &Out           = data[2].d; // output
```

I do the following:

```
#define UDATA(data)                                                          \
  const double &In            = data[0].d; /* input */                     \
  const int    &SomeAttribute = data[1].i; /* input parameter */           \
  double       &Out           = data[2].d; /* output */
```

Note that I change input parameters to `const&`. This ensures that I don't change the input values and eliminates the stack copy of the variables. (Hey, a cycle saved is a cycle earned. OTOH, any decent optimizing compiler would eliminate unused stack variables….) If I need local copies of input variables, I can declare them myself.

As you can see, it's a bit of effort to use the `UDATA()` thing, changing the comments (or deleting them), adding the trailing continuation character ("\"), and whatnot. So, you might ask: Is this `UDATA()` thing is really necessary?

Well, no. Absolutely not. But I've messed up more than once so I'm sticking with this until I devise a better solution. If you don't like it, change the template.

The Compiler Support

I decided to implement separate headers for DMC, MinGW, and MSVC support (*_DM.h, *_MGW.h, and *_VC.h, respectively). The compiler is automatically detected and the correct headers are included by DbgLog.h and Cblock.h.

Currently, the MSVC and MinGW versions require C++17. This may change to C++20 in the future to, hopefully, use less pre-compiler stuff.

For other modern compilers, the *_VC.h versions may work if you remove/modify pre-compiler version tests.

Example Compile/Link Commands

DMC

```
"C:\program files\qspice\dm\bin\dmc.exe" -mn -WD Cblock.cpp kernel32.lib
```

MSVC

```
cl.exe /std:c++17 /EHsc /LD Cblock.cpp /link /PDBSTRIPPED /out:Cblock.dll
```

MinGW

```
C:\msys64\mingw32\bin\g++.exe -O2 -std=c++17 Cblock.cpp -shared -static -o Cblock.dll
```

Note: Thanks to QSpice forum member @Jope for cluing me in on the required MinGW compiler/linker options.

The Wrap-Up

Maybe you'll find the template(s) useful. Please let me know if you find problems or have improvements.

--robert