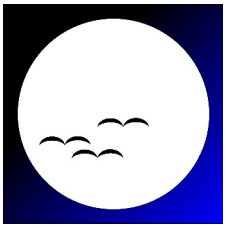

QSpice C-Block Code Generator (QCodeGen)

User Documentation

Document Revisions:

2024.02.13 – Initial document version.

2024.03.13 – Updated for v0.9.3.

QCodeGen code is based on potentially-flawed reverse-engineering the QSpice schematic file format. While I believe it to be reliable at the time of release, changes to QSpice may break it in the future.

Overview

QCodeGen is a C-Block component code generator for QSpice similar to the built-in QSpice code generator. Improvements over the built-in generator include:

- Code generation using custom user-defined code templates and keyword parameter substitution.
- Updating of existing `*.cpp` code for schematic component changes with minor programmer effort.
- Generation of code without overwriting existing `*.cpp` component code files.
- Selective copying of generated code fragments to the clipboard to paste into existing code.
- Improved error analysis for malformed port and string attributes.
- A size parameter for the passed `uData` array.

Installing/Uninstalling QCodeGen

The most current version of the QCodeGen source code and binaries are available on my GitHub QSpice repository: https://github.com/robdunn4/QSpice.

Installation

Binaries and Sample Code Templates

The QCodeGen executable, required library files, sample code templates, and documentation are provided in a `*.zip` file. There is no special installation program – simply unzip into a folder of your choosing.

Note that the executable (QCodeGen.exe) is compiled for 64-bit Windows. If you need a 32-bit version, you will need to compile the program from the sources.

Source Code

The source code was developed using the Qt framework (Qt Creator 12.0.2, Community (free) version) and MSVC 2022 (also free). You can get Qt at https://www.qt.io and Visual Studio at https://visualstudio.microsoft.com/. The Qt project file is included if you wish to compile the code yourself.

Uninstallation

There is no dedicated uninstallation program. Simply delete the installation folder and sub-folders from wherever you installed them.

If you want to remove all traces of QCodeGen:

- QCodeGen saves configuration data (preferences/settings) in the user's home directory. The full path to the `QcodeGen.ini` file is available in the Help | About dialog box.
- I believe that Qt applications automatically save some general configuration information in the Windows Registry, stuff like the last folder opened in a file dialog. I could be wrong. Anyway, you'll need to see the Qt site for more information if this is an issue.

License

Using QCodeGen

Getting Started

Using QCodeGen is straight-forward. Open a QSpice schematic that contains at least one C-Block component. The available components are displayed in the top tab bar. The bottom tab bar switches between a summary view and a code-generation view for the selected component and active code template.

QCodeGen error-checking is more complete than that of the QSpice code generator. The summary view lists component details and, importantly, any errors detected. The included `Error_Tests.qsch` schematic includes three components with deliberate errors for demonstration purposes. The X1

component contains many errors; the X2 component corrects the errors in X1; and X3 demonstrates incomplete pin data-type and I/O selections.

You cannot edit the Summary or C-Block Code text in QCodeGen. You can, however, copy the text to the clipboard using `CTRL-C` or the Copy button. If any text is selected, the Copy button copies only the selected text to the clipboard. Otherwise, all text is copied.

The Save button opens a dialog to select a location and name for saving the displayed text.

Use the Load Template button to select a different code-generation template.

If you edit the schematic or template file while QCodeGen is open, you can use the Reload button to reload the schematic & template from disk. QCodeGen immediately reparses both and updates the displayed Summary and C-Block Code contents.

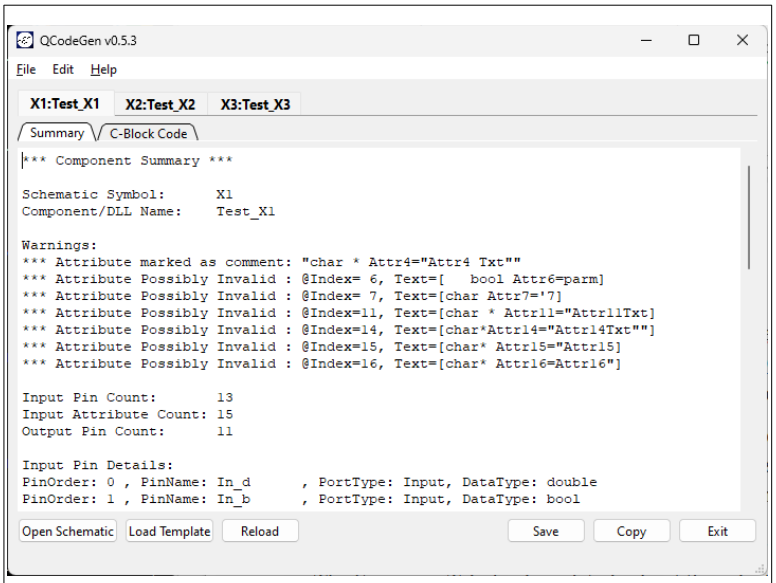

Figure 1: Error_Tests.qsch Loaded

Preferences Settings

The Preferences Settings dialog lets you configure:

- The default initial folder location for loading schematic files.
- The code template (`*.qcgt`) to load at startup.
- Text substitutions for code generation (author and user-defined fields).

See the Custom Code Templates topic below for details about creating your own custom code templates.

What QCodeGen Doesn't Do

Selective Code Section Generation

The QSpice code generator provides a dialog to select partial code generation. This seems unnecessary and would complicate templates so QCodeGen doesn't include this feature directly. Simply include all of the code features in your templates and, if you don't need them, comment them out during development and delete them entirely when the code is finalized.

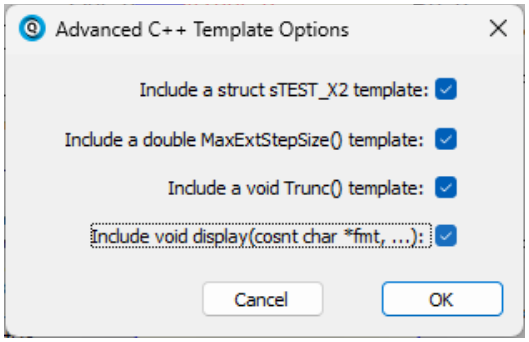

Figure 2: QSpice Selective Code Generation Dialog

Custom Code Templates

Custom code templates are the key purpose of this tool. You can define a code template and QCodeGen will replace certain text patterns with code/text to generate your custom C-Block code. You might have different templates for specific C/C++ compilers, versions that include debugging features or enable logging to files, or, well, anything that you might find useful.

Code template files can be created with any plain-text editor. The default file extension is "`.qcgt`".

Text Substitution Patterns

When generating code, QCodeGen replaces certain text patterns in the template with code. The patterns use this format:

```
%%keyword%%
```

Where "`keyword`" is one of the values in the table below.

There are two types of keyword patterns: *Embedded* and *Block*.

Embedded Keyword Patterns

Embedded keywords may appear anywhere within the template. That is, they can appear anywhere on a line along with other text. The associated pattern text simply replaces the entire pattern in place.

Block Keyword Patterns

Block keywords are replaced with (potentially) many lines of code. They must appear on a line by themselves. That is, spaces before and after the keyword pattern are permitted but no other text is allowed. If a block keyword pattern appears in a line that contains other non-white-space text, they are simply ignored and remain as-is in the generated code.

If a block keyword pattern is indented, the each line of replacement text will also be indented.

Keyword Patterns

The following keywords are currently recognized and replaced with relevant text by QCodeGen. The keywords are shown in mixed case ("camel-case") but the pattern matching is case-insensitive.

Keyword	**Type**	**Replaced With**
`Author`	Embedded	Author name as configured in the Preferences Dialog.
`AppName`	Embedded	Application name ("QCodeGen").
`AppVersion`	Embedded	Application version, e.g., "v0.5.3".

`User1`
`User2`
`User3`
	Embedded	User-defined text as defined in the Preferences Dialog. Could be used for comments, copyright notices, log file names, pre-processor directives/values or, well, whatever you think of.

Keyword	Type	Replaced With
`DateShort` `DateLong`	Embedded	Code generation date in short or long format, e.g., “2024.03.10” and “Sun Mar 10 2024” respectively.
`DateTimeShort` `DateTimeLong`	Embedded	Code generation date/time in short or long format, e.g., “2024.03.10 17:46:38” and “Sun Mar 10 17:46:38 2024” respectively.

`ComponentName`
`ComponentNameLC`
`ComponentNameUC`
	Embedded	The component name from the Symbol Properties “`1st Attribute`” text in original, lower, and upper case respectively. `ComponentNameLC` must be used for the evaluation function name.

`ComponentDesc`	Embedded	The component description taken from the Description field in the Symbol Properties.
`UdataSize`	Embedded	Number of elements in the `uData[]` array passed into the evaluation and `Trunc()` functions. This value is determined by the number of input/output ports and attributes when QCodeGen parses the schematic file.
`Undef`	Block	“`#undef`” pre-processor directives for each of the `uData` input/output port variable names.
`UdataAll`	Block	`uData[]` array element declarations. Typically used in the evaluation function.
`UdataOutput`	Block	`uData[]` array elements for output ports only. Intended for use in the `Trunc()` function.
`UdataSaveOutput`	Block	Code for temporary copies of `uData[]` array output port elements. Intended for use in the `Trunc()` function.
`UdataRestoreOutput`	Block	Code to restore `uData[]` array output port elements from temporary copies. Intended for use in the `Trunc()` function.

Included Templates

The QCodeGen installation files include several sample templates in the templates sub-folder. As of this writing, the included templates are:

Template File	Description
`qspice_default.qsch`	Very close to the QSpice auto-generated code. Same function and structure names and basic code structure.
`qspice_better.qsch`	A slightly improved version of the QSpice auto-generated code. Unnecessary struct keywords removed, instance data struct renamed to `InstData`, evaluation function slightly changed, `malloc()` replaced with `calloc()`, and the `bzero()` function removed.

Template File	Description
`update_blocks_only.qsch`	This template is useful if you changed the C-Block in the schematic and want to generate only the major bits that change to copy/paste into existing code.
`text_subst_test.qsch`	This template contains all of the keyword patterns for testing/checking QCodeGen pattern substitution. Maybe useful if you're checking your configured preferences or modifying/testing the source code.

Additional templates will be available shortly.

Refreshing *.cpp Code

If you make changes to a component's ports or attributes, you'll need to regenerate the block code sections.

QCodeGen makes this reasonably easy. When generating code, the program leaves a commented version of the block keyword pattern in the code. For example, "`%%UdataAll%%`" is replaced with:

```
/* %%UdataAll%% */
[ ...the replacement text block code... ]
```

To refresh the `*.cpp` code:

1. For each block section, delete the code created previously by QCodeGen.
2. Uncomment the block keyword.
3. Save the `*.cpp` file.
4. Load the `*.cpp` file into QCodeGen as a template.
5. Open the schematic.
6. Save the C-Block code generated by QCodeGen.

That's pretty simple and less error-prone than the QSpice cut/paste alternative.

Conclusion

I hope that this program will make life as a QSpice C-Block component developer just a bit easier. Let me know if you find problems or have suggestions to improve the program or documentation.

--robert

Appendix A

Error Analysis

The QSpice code generator will produce invalid code under certain conditions. In particular, a malformed string attribute will corrupt the uData pin/attribute definitions/offsets. For example, if the component contains the following malformed string attributes as viewed in the Symbol Properties pane, QSpice will silently generate invalid code.

Text Order	**Content**
2nd Attribute	`char* mystring1=”some text”`
3rd Attribute	`char* mystring2=”more text`
4th Attribute	`char* mystring3=”even more text”`

Because the mystring2 parameter isn’t properly closed, QSpice will not generate code for the mystring3 or any subsequent string attributes.

There are other ways to break stuff.

Note: While the warnings in the Summary Tab should prevent errors during component development, QSpice generates its internal version of the uData parameters and offsets each time the simulation is run. If a parameter is subsequently changed in the schematic and is malformed, the runtime uData array will not match the compiled code.

There is no easy fix for this since QSpice doesn’t pass a size parameter at runtime. In some cases, such problems would corrupt the netlist sufficiently to call attention to such problems. But in other cases, you won’t be so lucky.

One might consider adding a final end-of-array sentinel value. It would need to be the last defined output pin and checked at runtime. A better solution would be for QSpice to pass an element count when the simulation is run….