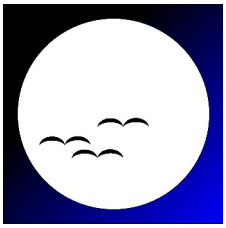

QSpice C-Block Components

Revisiting MaxExtStepSize()

Document Revisions:

2024.06.04 – Initial published version.

2024.06.08 – Rework and simplification.

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 fifth 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.***

This document revisits the MaxExtStepSize() function as a solution for a QSpice "peculiarity." 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 and code samples for this one.

Overview

A common C-Block component requirement is a clock source to trigger state changes. I thought that a QSpice pulsed voltage source as an input would be an ideal clock trigger for such components.

I was wrong.

This short paper describes the problem and implements a clock component using `MaxExtStepSize()`.

The Problem

One of the first QSpice C-Block components that I developed and shared was an audio WAV file generator. It would take two input signals from the schematic, sample, and write them to a stereo WAV file. To provide a clock source to trigger sampling at the desired WAV frequency/sample rate, I did something like Figure 1. Simple. Straight-forward.

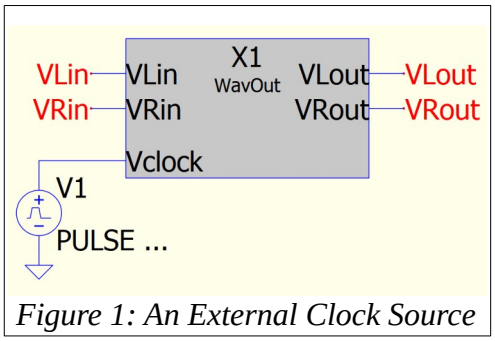

Figure 1: An External Clock Source

It worked just fine in my limited testing. Eventually, a user complained that the WAV file was dropping samples when longer simulations were run. This took a while to chase down because it really was related to the longer simulation run times. The V1 clock source was dropping pulses. Really.

But why?

The QSpice Adaptive Time-Step Algorithm

Note: I don't have any inside information so the following explanation is based solely on personal observation and conjecture.

QSpice is fast. Part of the speed magic is "adaptive simulation step selection." That is, QSpice analyzes the circuit behavior and adjusts the sample/time density dynamically. Simulation sample rate is increased around rapidly changing voltages or currents. Where voltages or currents change slowly, the sample rate is decreased.

I suspect that QSpice also gauges the impact of changing input voltages to output changes of schematic elements, a sort of sensitivity analysis for each component. Where changing inputs don't change outputs much, QSpice may reduce the weight of the element in the adaptive step selection algorithm.

All of that cleverness makes sense for well-defined circuit elements like native symbols and models. C-Block component behavior, on the other hand, isn't well-defined. In fact, to QSpice, the DLLs are black boxes.

I presume that the pulse signal in Figure 1 doesn't do anything that QSpice can clearly associate with "interesting" circuit activity (especially if the outputs are floating). As a result, QSpice would occasionally simply drop clock transitions from V1.

I needed a more reliable solution.

Note: The QSpice author has suggested that adding a capacitor across Vclock to ground would work around this behavior. I've not analyzed that solution.

A Self-Clocking Component

The `CBlockBasics5.cpp` code demonstrates one way to generate a clock internally using `MaxExtStepSize()`. It's pretty straight-forward.

The evaluation function does all of the heavy lifting. After some one-time initialization, it calculates and saves the next clock tick time and time increment in per-instance data (`next_t` and `incr_t`, respectively). An instance variable (`tickCnt`) keeps up with the click tick count; `calcTickTime()` uses it to calculate the next clock tick time with minimal rounding error for minimal clock jitter.

When a clock tick occurs, the time increment (`incr_t`) is set to `TTOL`. Otherwise, the increment is the time until the next clock tick.

`MaxExtStepSize()` simply returns the next step increment (`incr_t`) as set by the evaluation function. (The test for `incr_t <= 0` handles the several calls that QSpice makes during initialization.)

Note that `TTOL` sets the *maximum* rise/fall time for clock transitions. The actual transition time may be less. In fact, the edge transition time may vary from tick to tick.

So, that's it. It's pretty easy and, in my limited testing, more reliable than the schematic clock input.

Conclusion

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

--robert