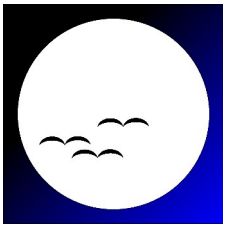

SPI Components For QSpice (SpiIO)

Developer Documentation

Overview

In an ideal world, you wouldn't need the SpiIO components – you'd have third-party components, both master and slave devices, that properly simulate actual SPI device behavior.

In this fantasy land, we'd have vendor-supplied micro-controller components that we could drop into a schematic. We'd write some actual micro-controller code and plug it into the component. We'd drop in some vendor-supplied slave device components, wire them up to the master, and Tada! We'd have excellent simulations of actual real-world devices in operation with near-zero sweat.

But we don't live in that world. At least not yet.

Until that day arrives, I've cobbled together some code, a framework of sorts to let us simulate "ideal" master and slave devices with minimum effort. In theory, you can do whatever you want/need to simulate actual master/slave devices.

Hopefully, this document will get you started.

> Note: If you are not familiar with SPI, what follows will almost certainly be unintelligible. You could start here: Wikipedia Serial Peripheral Interface.

Basic Features

SPI Modes

The SpiIO master and slave components support all 4 SPI modes (0-3) for the various combinations of SPI clock polarity (CPOL) and phase (CPHA). The mode for each component is set as an attribute in the schematic ("SpiMode" attribute) so recompiling the DLL is not necessary.

SCLK Speed

The SPI clock speed (SCLK) is also set with an attribute of the master device in the schematic ("SpiFreq" attribute) so recompiling the DLL is not necessary. There aren't specific SPI speed standards so look at the data sheets for whatever devices you're trying to simulate.

SPI Buffer Size

The code implements arbitrary buffer sizes up to 32-bits. In my limited experience, SPI devices typically have buffer sizes in multiples of 8 bits but you can specify any value for the buffer size (up to the 32-bits) in the code. If a larger buffer is needed, simply changing the internal buffer type from uint32_t to uint64_t and changing some guard code would be simple. Or you could modify the code to do what hardware typically does for extended data I/O – send as many 8-bit buffers as needed while keeping the CS line active.

Modifying SpiIO For Specific Devices

Master & Slave Devices

The code that handles sending/receiving data between master and slave devices isn't pretty. Fortunately, you need not understand it; generally, you just need to change what happens before and after devices exchange data. But you can also handle partial transmissions as explained below.

Within the code, there are three relevant functions where you can add code:

- `loadDataBuf()` – Called before the data exchange begins. Put data into the output buffer. (Master & slave devices)
- `processDataBuf()` – Called after the data exchange ends. Process data received in the input buffer. (Master & slave devices)
- `bitReceived()` – Called as each bit is received in the input buffer. (Slave devices only)

The last item might seem strange. It's here for a reason: Some more advanced slave devices must receive the first few bits before deciding whether they are being instructed to do something entirely internal to to the slave device (such as setting a power-on configuration) or being instructed to return some information (such as a stored configuration or active condition). See the Microchip MCP4261 for a complicated example.

I'll not try to describe this further. Just know that this "check some bits received and change what remaining bits will be sent" feature is easy to use when needed.

Master Devices

The master device code drives the SCLK so it is a bit more complex than the slave code. The demonstration code uses an input port ($\overline{\text{EN}}$) to trigger the data exchanges periodically. The code can be modified for some other triggering method.

Slave Devices

Some slave devices have only one of the MOSI/MISO lines. In fact, the ADC component doesn't do anything with data from the MOSI line so it could be omitted. Likewise, the MISO line could be omitted from the DAC component as the master does nothing with the received data.

Final Notes

I hope that you'll find the SpiIO components useful. Please do let me know of any issues. Or – better yet – figure out how to fix bugs and send me the corrected code. (I promise to give credit.)

You can find me on the QSpice support forum forums as @Rdunn.

My QSpice code/components are available on this GitHub page.

–robert