Getting Started with NI SPI API

Outline <Make these links>

Introduction

Requirements

API Overview

Using the API

Debugging

Customizing the API to Add 2 SPI Data Lines

Customizing the API to Have a Busy or Ready Signal

Introduction

The NI SPI API allows you to quickly get SPI communication up and running on your NI System on Module. This document will walk you through the process of setting up the SPI Engine and configuring it to work with your SPI module. It will also cover common troubleshooting techniques and ways you can further customize this API to work with other SPI configurations.

Requirements

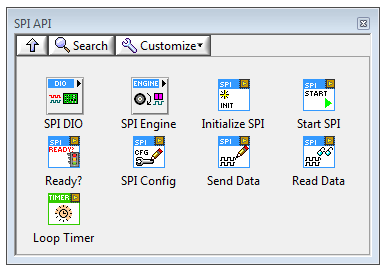
Software

* LabVIEW Full Development System
* LabVIEW Real Time
* LabVIEW FPGA

Hardware

* NI System on Module
* NI 9651
* Target SPI Module

API Overview

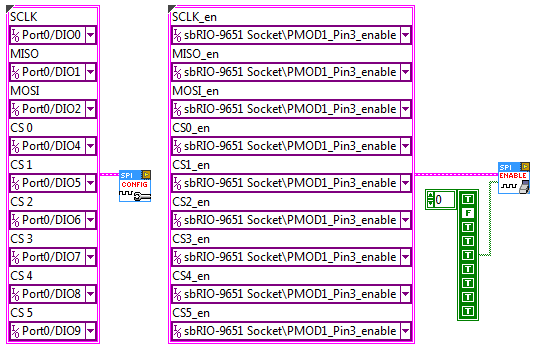


There are different components of the SPI API to control the communication between the FPGA and SPI module. The core component of the SPI API that controls the digital communication between the hardware components is the SPI Engine. The SPI Engine can be configured to use specific FPGA DIO lines to act as the MISO, MOSI, CS, and CLK lines for the SPI communication. The VIs in the SPI DIO sub-palette are used to configure which DIO lines will be used by the SPI Engine, as well as read and write to these DIO lines.

Alongside the SPI Engine, there are VIs available in the palette that are used to queue up commands to be sent out on the SPI bus. The SPI Engine uses FIFOs to receive these commands to write SPI messages out on the bus and then return messages read back from the SPI Slave(s).

* Select DIO lines

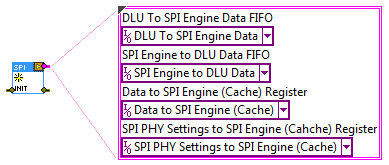
When first using the API, the initial step before starting the SPI communication is to setup the digital I/O lines that will be used for the data transmission. The Digital Config VI from the SPI DIO sub-palette can be used to select the lines. Placing this VI and creating a control for the ‘SPI DIO Lines’ input will allow specific DIO lines to be assigned to the different required SPI I/O lines, as shown on the left below.



* Configure enable lines

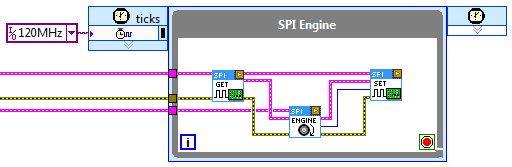
If you are using FPGA hardware that requires the DIO lines to either be enabled for output or disabled for input, such as the sbRIO-9651 used in the SPI API examples, you can use the Enable Digital VI from the same palette to set these enable states, shown on the right above.

* Initialize Memory Items



The Initialize VI sets up all of the memory items that the SPI Engine and Command Queuing VIs will use to communicate between one another.

* Setup SPI Engine

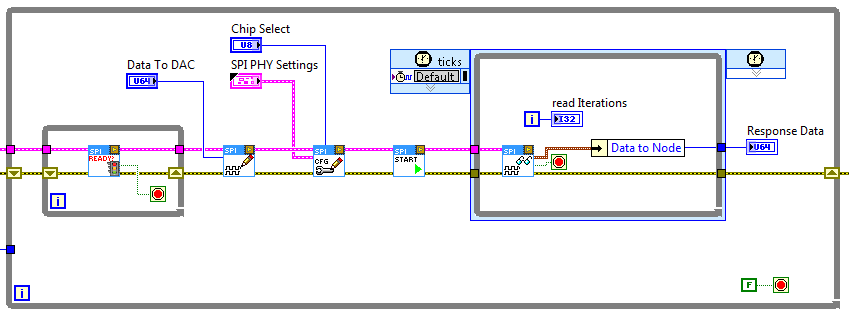


The SPI Engine runs within a Single-Cycle Timed Loop to read and write data to the DIO lines defined in the previously mentioned VIs. It uses the memory items created by the Initialize VI to both receive commands queued up in other loops (discussed in later section), receive data that needs to be written onto the SPI bus, and return data read from Slaves on the bus.

The first VI encountered in the SPI Engine above, the Read Digital Lines VI, reads the current state of the input SPI lines (MISO line) to pass into the SPI Engine. The next VI, the SPI Engine VI, is the actual engine that processes commands and controls the DIO lines for output. Finally, the last VI writes data to the output digital lines (MOSI, CS, and CLK) to communicate commands to Slave SPI devices.

* Setup Command Queuing loop

With the SPI Engine in place, the next step in using the API is to setup the Command Queuing loop where commands can be sent to the SPI Engine via the memory items created by the Initialize VI. In this loop we can choose to write out specifically formatted messages to our Slave SPI devices, and then read their raw responses.



The following discusses the purpose of each VI in the above image.

Ready.png The Ready VI checks to ensure that there is enough room to queue up new commands to the SPI Engine.

SPI Write.png The SPI Write VI takes a data set from the user that will be sent to the SPI Engine for transmitting on the MOSI line.

SPI Config.png The SPI Config VI takes configuration data from the user that describes how the SPI Engine should transmit the data provided by the SPI Write VI. The configuration options define the length of the data in bits to be transmitted, the Chip Select line to use, etc.

Start.png The SPI Start VI takes the information provided by the SPI Write and SPI Config VIs and transfers it to the SPI Engine.

SPI Read.png The SPI Read VI polls the SPI Engine to read the MISO response from the selected SPI Slave.

Using the API

The SPI API is easy to configure for any SPI module that you would like to communicate with. This walk through will use the MAX5216 DAC as an example, but will also explain how this same process could be done for other SPI chips. The example code discussed here is the s

Debugging