Interfacing the LabJack U12 to SPI Devices

Introduction:
This document explains how to use the LabJack U12's Synch function to interface with SPI devices. Synchronous (SPI) and asynchronous (RS232) communication opens up a lot of flexibility, but can also be rather complicated and hard to troubleshoot. The Synch and Asynch functions might not be as easy to use as other U12 functions, and we will not be able to provide extensive support for every problem with a particular untested interface device.

Note: The Synch function is included in firmware 1.1 and above.

Contents:

SPI Basics:
SPI, serial peripheral interface, typically uses four digital IO lines: a CS, data out, data in, and a clock. SPI is often called a 3-wire or 4-wire synchronous interface.

The CS (chip select) allows multiple devices to share the same data and clock lines. Each device on the SPI bus will have its own CS line. Normally CS operates as !CS (not CS), this means that the !CS line is set unless communication with a chip is in progress. A chip is selected and read from / written to while its !CS line is low. When the !CS line returns high, data is latched into the chip's register.

SPI is a versatile protocol, which allows for several modes of operation. The modes specify the clock line polarity and when data is sampled.

Mode 0: The clock line is normally low and data is sampled on the rising edge of the clock. (a.k.a. mode A)
Mode 1: The clock line is normally low and data is sampled on the falling edge of the clock. (a.k.a. mode B)
Mode 2: The clock line is normally high and data is sampled on the falling edge of the clock. (a.k.a. mode C)
Mode 3: The clock line is normally high and data is sampled on the rising edge of the clock. (a.k.a. mode D)

Circuit considerations:

- All D lines to be used should not have large series resistance. If a CB25 is used the shorting jumper should be installed on all SPI lines.
- 1nF capacitors should individually couple the data and clock lines to ground.
- Devices that utilize 3 wire SPI combine the data in and data out lines. The LabJack U12 can be connected to 3 wire devices by placing a resistor in series with the MOSI line, then tying both MISO and MOSI to the data pin on the device.

The Synch function:

The synch function performs 3 and 4 wire SPI communication, by manipulating the LabJack U12's digital IO lines.

IO lines used for SPI:

MOSI (Master Out Slave In) is D13.
MISO (Master In Slave Out) is D14.
SCK (Serial ClocK) is D15.
CS (chip select) optionally chosen as D0-D7.

Several parameters control the way the LabJack U12 performs SPI communication.

Returns:	Error codes. The error codes should be checked each time a LabJack function is called. Codes can be interpreted using the GetErrorString function.
idnum:	ID number of the LabJack. –1 = first found.
demo:	Greater than 0 allows the function to be called without a U12 connected to the computer.
mode:	Synch supports four modes of SPI communication. The four modes control the default state of the clock line and the edge that data is valid on. The following describes the four modes. 0: Mode A: The clock line is normally high and data is valid on the rising edge of the clock. 1: Mode B: The clock line is normally low and data is valid on the falling edge of the clock. 2: Mode C: The clock line is normally high and data is valid on the falling edge of the clock. 3: Mode D: The clock line is normally low and data is valid on the rising edge of the clock.
msDelay:	If >0, a 1 ms delay is added between each bit.
husDelay:	If >0, a 100 us delay is added between each bit.
controlCS:	If >0, the Synch function will handle chip select.
csLine:	The D line to be used as CS (0-7).
csState:	0 = normal !CS 1 = CS

configD:	If >0, the directions of the IO lines D13, D14, D15, and !CS are configured.
numWriteRead:	This is the number of bytes to send/receive.
data:	A array of 18 longs. This array contains data to be sent and is loaded with data received. When sending data, index 0 of the array will be sent first. Data received is stored in data starting with index 0. All unused locations in the array must contain 0.

<u>Examples:</u>

DAC7611:
The DAC7611 is a 12-bit digital to analog converter that receives data via an SPI bus. It has an internal reference, so the output voltage ranges from 0 to 4.095 volts and does not vary with supply voltage.

The DAC7611 adds the unusual feature of a LD (latch data) line. A pulse on LD informs the converter to update the output to the most recently received 12 bits. The LD is normally high. Due to the presence of the LD line, the CS line is not necessary to run the DAC7611. If CS is tied low the DAC will read all data on the bus and update the output voltage when a LD pulse is detected. However, the CS is necessary if more than one chip will share the data and clock lines. The Synch function does not support the LD, a LD pulse will be created using EDigitalOut.

Another consideration is that the DAC7611 needs 12 bits of data while the LabJack can only send multiples of eight. This does not pose a problem because the DAC7611 will ignore all data with the exception of the most recent 12 bits.

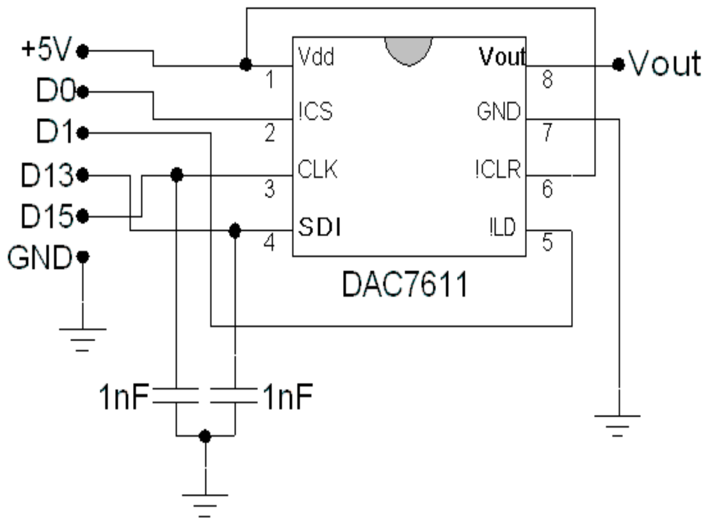
An example written in C++ can be found in the SPI example files: DAC7611.cpp

Hardware setup:
This example uses four D lines to interface with the DAC7611.
D lines used for this example:

MOSI D13
SCK D15
!CS can be D0-D7. This example uses D0.
LD can be any unused D line. This example uses D1.

The MISO line is not used because the DAC7611 only receives data.
1nF capacitors are installed on both the data and clock lines. These capacitors should be adjacent to the DAC7611.

Example schematic to connect a DAC7611 to the LabJack U12:



Software setup:
The software controlling the LabJack U12 must load the data array, call the Synch function, then use EDigitalOut to create an LD pulse.

The data to be sent needs to be converted into bytes and placed in the proper elements of the data array. For the DAC7611 a number between 0 and 4095 (12 bits) will need to be split into two bytes. One byte will contain the upper 4 bits and another byte will contain the lower 8 bits. The conversion can be accomplished using an integer divide and a remainder divide. The Integer divide only returns the whole number result. While the remainder divide only returns the remainder of the division. Once the data has been converted it must be loaded into the proper positions in the data array. The first element of the data array will be the most significant byte sent by Synch. Bytes from the array are then sent in ascending order. The most significant byte is the result of the integer division and should be placed in the first element of the data array. When Synch is called two bytes will sent to the DAC7611. The upper four bits of the most significant byte will be ignored, leaving 12 bits of data in the DAC's shift register. The following two lines of C code will split bVolts (binary representation of the desired output voltage) into two bytes and place them into the data array.

```
data[0] = bVolts / 256;         // Integer division (# of 256's)
data[1] = bVolts % 256;         // Remainder division
```

To send the data, each parameter for the Synch function must be determined:

idnum:	= -1 (first LabJack found)
demo:	= 0 (demo mode off)
mode:	The datasheet for the DAC7611 describes a SCK which is normally high with data valid on the rising edge of the clock pulse. This corresponds to mode = 0 (mode A).
Delays:	The DAC7611 requires a minimum of 60 ns per bit. The LabJack U12 takes 6.25 us to send a bit, so no delays are necessary.

	msDelay = 0 and husDelay = 0.
controlCS:	= 1 to allow the Synch function to control CS.
csLine:	= 0 This example uses D0 for CS.
csState:	= 0 The DAC7611 uses a standard !CS line.
configD:	= 1 The Synch function will set the direction of the D lines.
numWriteRead:	= 2 The DAC7611 requires 12 bit of data.

After sending the data, a LD pulse must be sent before the DAC7611 will update the output voltage. Two calls to EDigitalOut will create the LD pulse. The LD line should be set high prior to sending data.

```
EDigitalOut(&IDnum,demo,1,1,0);
EDigitalOut(&IDnum,demo,1,1,1);
```

Once LD has been pulsed the DAC7611 will update its output voltage.

ADS7813:
The ADS7813 is a 16-bit analog to digital converter that uses SPI to report the conversion result.

In addition to the SPI communication a !CONV line is used to start a new conversion. Conversion is initiated by pulsing the !CONV.

An example written in C++ can be found in the SPI example files: ADS7813.cpp

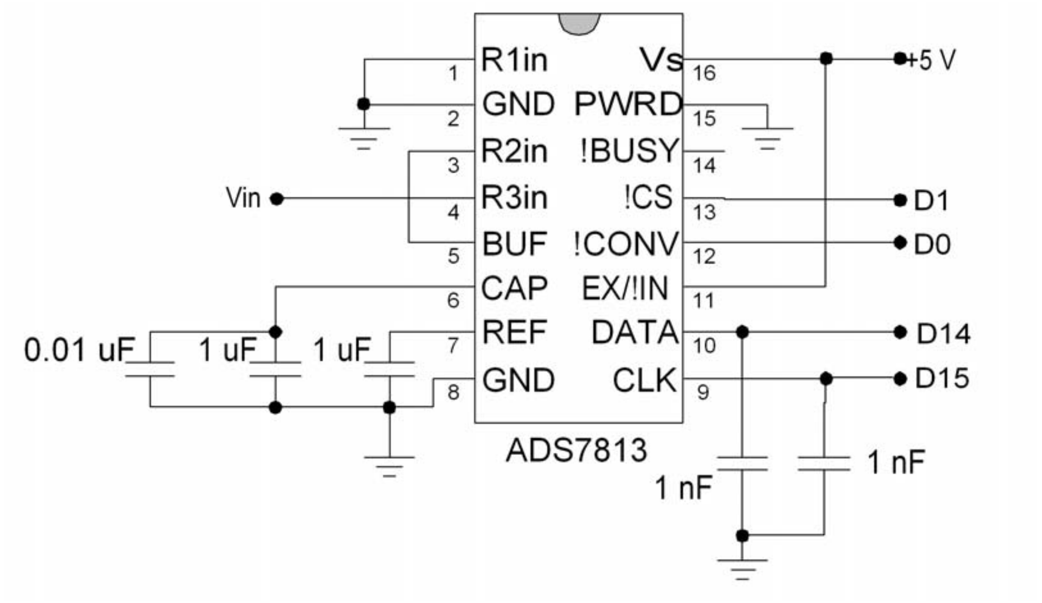
Hardware setup:
Four IO lines are needed to communicate with the ADS7813.

D0 !CONV
D1 !CS
D14 MISO
D15 clock

MOSI is not used because the ADS7813 only transmits data.

The test setup used +-5V input. Other input ranges require different connections to R1in, R2in, and R3in. The ranges and necessary connections can be found in table IV of the ADS7813 datasheet.

Example schematic to connect an ADS7813 to the LabJack U12:



Software setup:
The software controlling the ADS7813 must create the conversion pulse, allow 20 us for the conversion to complete, then read the result. The conversion pulse can be created using the EDigitalOut function. The result will not be ready until 20 us after the !CONV pulse. EDigitalOut requires ~20 ms to execute, which prevents the need for any sort of delay. After the conversion pulse the synch function can be used to retrieve the data.

For the synch function to retrieve the result of the conversion, all parameters must be set appropriately.

idnum:	= -1 (first LabJack found)
demo:	= 0 (demo mode off)
mode:	The datasheet for the ADS7813 indicates that the clockline should normally be low with data valid on the rising clock. This corresponds to mode 3 (D).
Delays:	The ADS7813 requires a minimum of 1.1 us per bit. The LabJack U12 takes 6.25 us to clock a single bit, so no delays are necessary. msDelay = 0 and husDelay = 0.
controlCS:	= 1 to allow the Synch function to control CS.
csLine:	= 1 This example uses D1 for CS.
csState:	= 0 The ADS7813 uses a standard !CS line.
configD:	= 1 The Synch function will set the direction of the D lines.
numWriteRead:	= 2 The ADS7813 creates a 16 bit result.

After synch has executed, the upper byte of the result will be in data[0] and the lower byte will be in data[1]. The two bytes are combined into a single number by multiplying data[0] bye 256 and adding to data[1].