**Arduino Library for Analog Devices’ AD57X4 Line of Digital-to-Analog Converters**

**Operation Manual**

Authored by Wesley Kuegler, NIFS Summer 2017 Intern at NASA’s Langley Research Center

**Introduction**

The AD57X4 library was originally sourced from a project by “Laser-Lance” at <https://www.laserlance.com/projects/arduino-dac-library-and-shield/>. In order to use it for the MEDLI2 SSE Sensor Simulator Development project, some additions had to be made. The library was reasonably robust but still incomplete, and some features were not implemented correctly. Further, it was built for single DAC operation, so adding daisy-chain DAC operation took some time. A full list of the changes made to the AD57X4 library can be found in “Changelog for Laser-Lance's AD57X4 Library.”

This library was altered specifically for use with an Arduino Micro running the sketch called “serial\_parser.” This sketch runs a text menu that can be operated over the appropriate serial port. Operation of the text menu is covered in “MEDLI2 SSE Sensor Simulator Arduino Firmware Operation Manual.” That document, along with the code for “serial\_parser.ino” itself, is very informative as an example of how to use the AD57X4 library.

In its current state, the AD57X4 library does not implement quite all AD57X4 DAC functionality. The most glaring omission is readback functionality for the DAC registers. Readback functionality existed for single DAC operation in the original version of the library, but time constraints prevented it from being implemented for daisy-chain DAC operation. Some values in the library are hard-coded in a configuration specific to the implementation for MEDLI2 SSE Sensor Simulator Development, but should be altered easily enough for other projects. All parts of the library currently working for daisy chain DAC operation should also work for single DAC operation.

**Operation Overview**

With the AD57X4 library in its current state, a user can use functions to:

1. Configure any number of daisy-chained DACs for 4-channel operation on the +/-10V range
2. Power on any number of configured, daisy-chained DACs
3. Properly format voltage set commands (for the +/-10V range) and push them out to every DAC in the daisy chain
4. Push a load command to every DAC in the daisy chain
5. Clear every DAC in the daisy chain
6. Push any valid DAC command out to the next DAC in the daisy chain
7. Push the SYNC line high to latch the command in the input shift register of every DAC in the daisy chain simultaneously

**The Constructor**

The constructor takes 6 arguments: “numChans1,” the number of channels per DAC; “numDACs1,” the number of DACs in the daisy chain; “volt,” an integer corresponding to a switch case in ConfigDACs() and pushDACvoltage() that selects the channel voltage range; “sync,” the number of the Arduino pin connected to the DAC SYNC line; “clr,” the number of the Arduino pin connected to the DAC CLR line; and “ldac,” the number of the Arduino pin connected to the LDAC line. Inside the constructor, the arguments are assigned to the class’ data members, the pins are initialized to output mode, the serial interface (SPI) is initialized, and calls are made to ConfigDACs() and PowerDACs().

**Configuring the DACs with ConfigDACs()**

ConfigDACs() configures all DACs in the daisy chain for operation. It sends the command “0x19000E” to each DAC, then latches the command. This command enables TSD and the SDO line, and sets the CLR select mode to negative full scale.

A large switch statement, “switch(numChans)” takes up most of the function. Originally, this function set “output” to the correct configuration command depending on the number of channels and the desired voltage range; however, the switch statement was never modified to work with a daisy chain of DACs. Instead, a group of four for loops at the end of the function is hardcoded to send the correct configuration commands for a daisy chain of 4-channel DACs on the +/-10V range. A call to SYNCdata() latches the commands. ConfigDACs() works for any number of DACs. The switch statement would be commented out, but for reasons unknown doing so appears to prevent ConfigDACs() from working properly.

**Powering the DACs with PowerDACs()**

PowerDACs() sets “output” to the correct power-on command depending on the number of channels. A for loop at the end of the function sends the power-on command to every DAC in the daisy chain. A call to SYNCdata() latches the power-on commands.

**Setting Voltages with pushDACvoltage()**

pushDACvoltage() takes two arguments: “DACVoltage,” the voltage to set the DAC channel to, and “DACNumber,” the number (1-4) of the channel being set. Passing 5 for DACNumber will set all 4 channels to the same value. The function uses a switch statement (“switch(voltSwitch)”) to convert DACVoltage to the correct bit pattern, depending on the selected voltage range. It then uses a second switch statement (“switch(DACNumber)”) to address the correct DAC channel. Finally, a masking operation converts “data” from a twos complement representation to offset binary before sending it using SendData(). This mask should be removed to operate the DACs in twos complement mode.

In order to set voltages for a daisy chain of DACs, pushDACvoltage() calls must be made in the correct order. A set command intended for the last DAC in the chain must be pushed out first, and conversely a set command intended for the first DAC in the chain must be pushed out last.

There are only two options when selecting which channels to set on each DAC: either all 4 channels will be set simultaneously, or only one channel will be set. When operating a daisy chain of DACs, each of the four channels should be set simultaneously for all DACs. For example, if the goal is to set all four channel values for each DAC in a three-DAC daisy chain, then a set command for channel A should be pushed in turn to each DAC in the chain, and LoadDACs() should be called to set a voltage for every channel A in the daisy chain. A set command for channel B should then be pushed in turn to each DAC in the daisy chain, and LoadDACs() should be called to set a voltage for every channel B in the daisy chain – and so on. To set all four channels in this hypothetical daisy chain, then, would require a loop to push three voltage set commands to the chain and then call LoadDACs(). This loop would need to run four times, once for each channel.

**Loading DACs with LoadDACs()**

This function pushes a load command to every DAC in the daisy chain, sets the LDAC pin low, makes a call to SYNCdata() to latch the commands, and sets the LDAC pin back to high. It can only load a voltage into the DACs if voltage commands have been properly pushed out to the daisy chain using pushDACvoltage(). Calling LoadDACs() will cause each DAC in the chain to simultaneously set their channels to the value previously pushed out to it using pushDACvoltage().

**Clearing the DAC Channels with ClearDACs()**

Using ClearDACs() to clear every DAC channel is very straightforward – simply call the function. It pushes the clear command to every DAC in the daisy chain, then makes a call to SYNCdata() to latch the commands. All channels will clear negative full scale, which is -10V for the original implementation. The clear value is also configurable to 0V; it can be set to 0V in ConfigDACs() by altering the “0x19000E” bitstream.

**Pushing Commands to the DACs with SendData()**

SendData() takes one argument: “data,” the 24-bit stream to be sent to the first DAC in the daisy chain. Programming subsequent DACs in the daisy chain requires that the commands be pushed through the input shift registers of the preceding DACs. For example, placing a command in the fourth DAC in a daisy chain requires that the desired command be pushed out using SendData(), and three more valid commands pushed out after to move the desired command to the fourth DAC in the daisy chain. SendData() uses the SPI serial protocol to move commands out through the Arduino’s MOSI pin.

**Latching Commands with SYNCdata()**

SYNCdata() only does one thing: it sets the SYNC line high for all DACs in order to latch the commands in the input shift registers of those DACs. This was made into a separate function – originally, the library simply set the SYNC line high inside SendData() – in order to allow for daisy chain operation. In daisy chain mode, commands should only be latched once a command has been pushed out for every DAC in the daisy chain.

**Implementing Readback Functionality with ReadDACs() and ReadDACsRegister()**

Time constraints prevented readback functionality from being implemented for daisy chain DAC operation. However, the framework for implementing that functionality was created, and is currently left commented out. In the original library, the function ReadDACsRegister() was used to address and read one of several registers for a single DAC. The intention in this updated library was to control ReadDACsRegister() with a new function, ReadDACs(), that would accept an array of byte pointers as an argument, pass that array to ReadDACsRegister(), and fill it with the read values from the DAC registers.