ELECOMP Capstone: Vicor 2019-2020 Project - LoCoDes
FSU API Documentation

Hardware required:

- Arduino Due
- Raspberry Pi 3+
- Micro-USB to USB cable (connects Arduino to RPi, as well as Arduino to PC to load code onboard)
- FSU System (incl. MUX card, DUT board, DAC81408, Waveshare AD/DA board)

Software required:

- Arduino IDE (installed on desktop system) - not needed if Arduino is already set up
- Python 3 (installed on RPi)

Setup Process:

1. Clone the following Github repository: https://github.com/csanzo/Capstone2020
2. If the Arduino board is already set up with the proper code loaded on board, skip to step 7.
3. In the Arduino IDE, go to Tools > Board > Boards Manager. A window will pop up. In the search bar, type "Due" and hit enter. Install the package that is listed (should read something like "Arduino SAM Boards").
4. Navigate to the folder on your system where Arduino libraries are saved (by default, this is usually located at C:\Users\[USER]\Documents\Arduino\libraries). Copy the contents of the "Arduino\libraries" folder from the Github repo into this folder. This will include the code to use the DAC81408, ADS1256, the FSU API, and an i2cdetect library.
5. Open the "Arduino\API_PiControl" folder from the Github repo. Open the .ino sketch file from this folder into the Arduino IDE. This is the code that allows the FSU API to be run and controlled from the Raspberry Pi. Note that there are two other sketches provided on the GitHub; "dac81408_example" and "ads1256_example". These are provided for debugging purposes, should you have issues with the DAC or ADC.
6. Load the "API_PiControl" sketch onto the Arduino Due. To do so, plug the Arduino in via a micro-USB to USB cable. In the Arduino IDE, go to Tools > Board and choose Arduino Due (there are two port options; refer to pinout linked below in "References" section to see which port you are connected to). Then, go to Tools > Port and choose COM# (the number here will be different almost every time. There should generally be only one result in this menu). Also, make sure Tools > Programmer is set to "USBasp." Finally, go to Sketch > Upload (keyboard shortcut: Ctrl+U). Wait for the code to be loaded onto the board (the final message will read "CPU reset.").

7. With the code successfully loaded onto the Arduino Due, make sure the Arduino is now connected to the Raspberry Pi via the programming port (this *must* be connected via the programming port in this step - if you need to update and reload code onto the Arduino, you can use the native port to do so simultaneously without disconnecting from the RPi).
8. Open a command prompt on the Raspberry Pi (shortcut: Ctrl+Alt+T) and type: "`ls /dev/ttyACM*`" then hit enter. The result that appears will be the port you will use to connect to the Arduino's Serial port. It will usually be '/dev/ttyACM0', but it is always good to double check.
9. To use the API, create a python file in the "Raspberry Pi" folder and include the following lines:

   ```
   "import vicorlib
   Device = vicorlib.Arduino('[Insert port from step 8]',
   9600)"
   ```

   You can now use API commands by referencing the "device" object (note that "device" is just a variable name, you can call it whatever you wish). Examples of this can be seen in the "api_mux_test.py" file from the Github repo. There are a number of other example files provided in this folder.

   You can run Python files by typing "python3 filename.py" in a command prompt. Be sure to type "python3" rather than just "python", as that would instead run Python 2 and the code will not run properly.

API Commands:

`select_pin(channel, pin, force=True)`

Sets the Force/Sense MUXes to select the specified DUT pin on the specified FSU channel.

Parameters:

`channel (int)` : FSU Channel to select pin on, 0 to 1

`pin (int)` : Pin number on DUT to select, 1 to 48

`force (bool)` : Optional, flag to set whether to activate the force MUX (sense is always active). Default value: True

Note: Since there are currently no sense MUXes, the sense functionality is not implemented.

Warning: Temporary change: Currently, a temporary 4-to-1 MUX is being used in place of the 48-to-1 MUX. As of this time, this command's "pin" argument will actually select the channel of the 4-to-1 MUX, with an accepted value of 0 to 3. More info below in the "Known Issues" section.

`select_resistor(channel, resistor)`

Sets the resistor MUX to select the specified sense resistor on the specified FSU channel. Resistors: 39k, 2k, 100, 5 (in order of MUX channel, i.e. selecting 0 will give the 39k resistor and 3 will give the 5 ohm resistor)

Parameters:

`channel (int)` : FSU Channel to select resistor on, 0 to 1

`resistor (int)` : MUX channel that desired resistor is on, 0 to 3

`force_voltage(channel, voltage)`

Forces voltage on a single FSU channel. The voltage forced here will go to the pin on the DUT set by `select_pin` on the same channel.

Parameters:

`channel (int)` : FSU Channel to force voltage to, 0 to 1

`voltage (float)` : Voltage to be set, 0.0 to 5.0V

Warning: The accepted voltage range is currently temporarily 0.5 to 5.0V. More info below in the "Known Issues" section.

`measure_voltage(channel)`

Measures the voltage on a single FSU channel. The voltage measured will be from the pin on the DUT set by `select_pin` on the same channel.

Parameters:
`channel (int) :` FSU Channel to measure voltage from
Returns:
The measured voltage, of type `float`

`force_current(channel, current)`

Forces current on a single FSU channel. The current forced here will go to the pin on the DUT set by `select_pin` on the same channel.

Parameters:
`channel (int) :` FSU Channel to force current to, 0 to 1
`current (float) :` Current to be set

`measure_current(channel)`

Measures the current on a single FSU channel. The current set here will go to the pin on the DUT set by `select_pin` on the same channel.

Parameters:
`channel (int) :` FSU Channel to measure current from, 0 to 1
Returns:
The measured current, of type `float`

`program_dut(address, payload)`

Sends payload of either 1-byte (8-bits) or 2-bytes (12-bits) to the specified address on the DUT.

Parameters:
`address (byte) :` Address on DUT to send payload to
`payload (byte OR byte list) :` The payload may be either a single byte, or a list of 2 bytes (for 12-bit payloads). This data will be sent to the specified address on the DUT

`read_dut(address)`

Reads data at specified address on DUT and returns the payload

Parameters:

`address(byte)` : Address to read payload from

Returns:

Payload read at the specified address, of type `byte OR byte list` (list for 12-bit read)

`extra_sense(adc_channel)`

This acts the same as the `measure_voltage` command, but runs off of the extra unused ADC channels, 6 and 7, which require a manual connection to DUT pins.

Parameters:

`adc_channel (int)` : Accepts values 6 or 7, ADC channel to read voltage from.

Returns:

The measured voltage, of type `float`

`i2cdetect()`

Performs an i2cdetect scan similar to Linux command of the same name. Outputs device table. For debugging purposes if having I2C issues.

Arduino pin connections to FSU system:

Note: DAC/ADC pinouts linked below in "References" section. All ADC pins refer to the Waveshare board's RPi header, and all DAC pins refer to SPI pins from the J8 header on the DAC81408EVM.

Warning: Connections in red indicate connections to be made for the 48-to-1 MUX setup. This setup is currently unavailable as an issue has caused damage to these boards. As a temporary band-aid fix, there is a 4-to-1 MUX setup, with connections highlighted in green below. When the 48-to-1 setup is back up and running, the necessary connections should be adjusted and this documentation should be modified to only include the 48-to-1 connections.

Connect Arduino 3.3V, 5V, GND pins to ADC 3.3V, 5V, GND pins.
Connect Arduino pins 11, 12, 13 to ADC pins 11, 12, 13, respectively.
Connect Arduino SPI pins (MOSI, MISO, SCLK) to both ADC and DAC equivalent SPI pins.
Connect Arduino pin 10 to ADC pin 15 (CS0 ADC chip select pin).
Connect Arduino pin 4 to DAC chip select (CS) pin.
Connect Arduino pins 20, 21, GND to the DUT's I2C SDA, SCL, GND pins, respectively.
Connect Arduino pins 22, 24, 26, 28 to FSU 0 force MUX card demux 23, 22, 21, 20, respectively.
Connect Arduino pins 30, 32 to FSU 0 force MUX card select pins S0, S1, respectively.
Connect Arduino pins 23, 25, 27, 29 to FSU 1 force MUX card demux 23, 22, 21, 20, respectively.
Connect Arduino pins 31, 33 to FSU 1 force MUX card select pins S0, S1, respectively.
Connect Arduino pins 47, 49, 51, 53 to FSU0 MUX ENA, ENB, SEL0, SEL1, respectively.
Connect Arduino pins 46, 48, 50, 52 to FSU1 MUX ENA, ENB, SEL0, SEL1, respectively.
Connect Arduino pins 14, 15, 16, 17 to FSU 0 resistor MUX ENA, ENB, SEL1, SEL0, respectively.
Connect Arduino pins 5, 6, 7, 8 to FSU 1 resistor MUX ENA, ENB, SEL1, SEL0, respectively.

Known Issues:

- There is a current known issue with the DAC where the voltage on any channel cannot go lower than 0.43V. There is a calibration algorithm that checks that the input voltage is actually what is being output, however if you input a value lower than 0.43V, it will never be output correctly due to this issue, causing the calibration to endlessly run since that condition is never met. For this reason, the accepted range for the voltage argument is actually 0.5V to 5V, temporarily, until this is fixed.
    - Note: problem may be specific to the chip currently in use, as it was known to be able to set the voltage all the way down to 0.0V in the past (last known time of this was when we were using a different chip/EVM board)
- There has been damage done to the 48-to-1 MUX boards, so we are temporarily using a 2-channel 4-to-1 MUX board. Once the 48-to-1 MUX boards are working again, please change this documentation to reflect that. As well, there is code in both the "vicorlib.py" file and "libraries/API/API.cpp" files that will need to be modified. The 48-to-1 MUX code has been commented out, and the temporary 4-to-1 MUX code has clear comments indicating where this code is. When it is time, please uncomment the 48-to-1 MUX code and delete the 4-to-1 MUX code where necessary.

Future Work:

There is some work that would need to be done in the future in regards to the current known issues; once the DAC issue is resolved, the accepted minimum value within the code should change from 0.5 to 0.0; this can be done by changing the "VMIN" global variable in the vicorlib.py file. As mentioned above, when the 48-to-1 MUX board issue is resolved, there are commented blocks of code for the 48-to-1 MUX in both vicorlib.py and libraries/API/API.cpp (quickly find by searching for "FIXME" in the files) that should be uncommented, and the corresponding 4-to-1 MUX code should be deleted (these will be clearly indicated with comments).

Other than these, one important thing to note is the difference in the LMH6321 board currently being utilized and the FSU PCB developed during the 2019-2020 Capstone project. Currently, only a single DAC channel is used to fully control each LMH6321. The LMH6321 will eventually be changed out for the PCB which makes use of two DAC channels. Therefore, some code will need to be developed in order to implement this functionality. Some existing code may also need to be changed, namely the four force/sense API commands. This is just a reminder that it will not be a simple plug-and-play type of swap out for the LMH6321 to PCB transition.

References:

- Arduino pinout: https://store.arduino.cc/usa/due Click the “Documentation” tab. The “Input and Output” and “Programming” sections under this tab will probably be the most relevant.
- DAC81408 pinout: https://www.ti.com/lit/ug/slau778a/slau778a.pdf Table 6 on page 7 has relevant information for SPI header pinout.
- Waveshare ADC pinout: https://www.waveshare.com/high-precision-ad-da-board.htm Note: the header pin numbers matches that of a Raspberry Pi.