User Manual

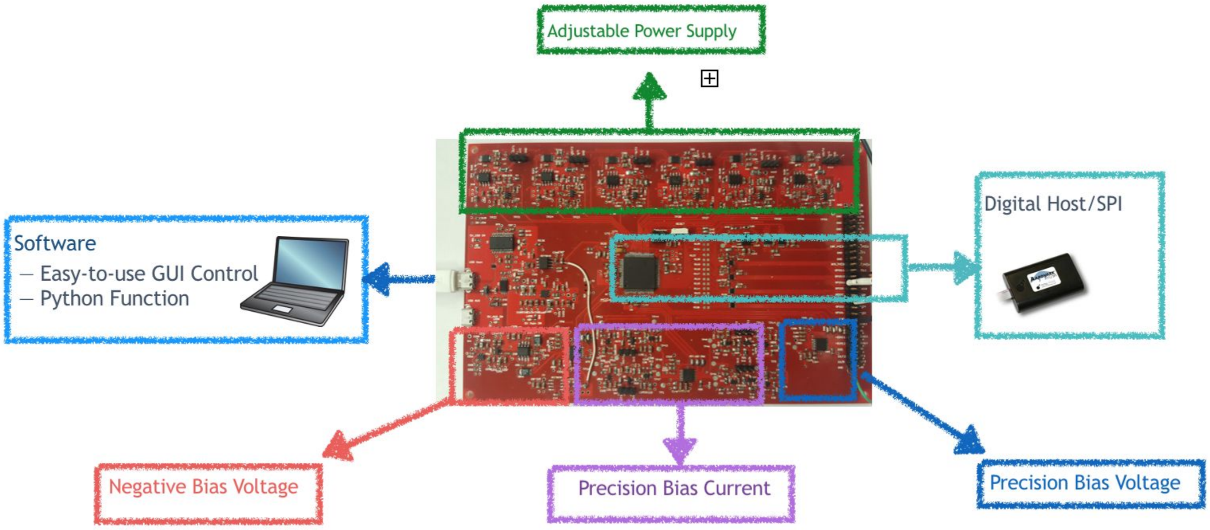
Evaluation Platform for Analog Integrated Circuits

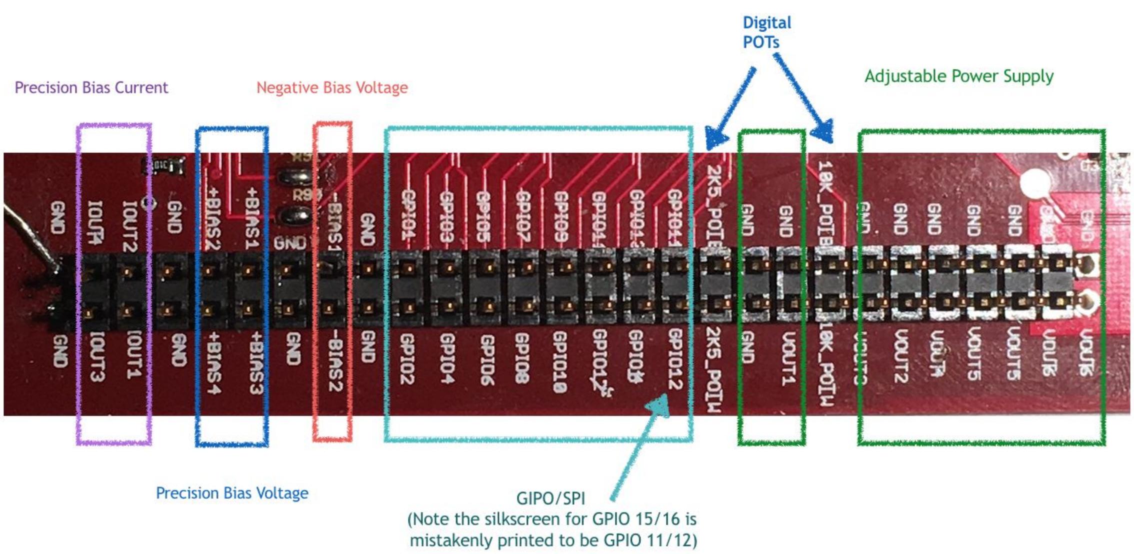
Eddy Zhen Zhang
Xirui Maria Xie

Date:
April 14th, 2017

Table of Contents

1 Overview


Adjustable Power Supply
Software
— Easy-to-use GUI Control
— Python Function
Digital Host/SPI
Negative Bias Voltage
Precision Bias Current
Precision Bias Voltage


Digital POTs
Precision Bias Current
Negative Bias Voltage
Adjustable Power Supply
Precision Bias Voltage
GIPO/SPI
(Note the silkscreen for GPIO 15/16 is mistakenly printed to be GPIO 11/12)

2 Python Functions

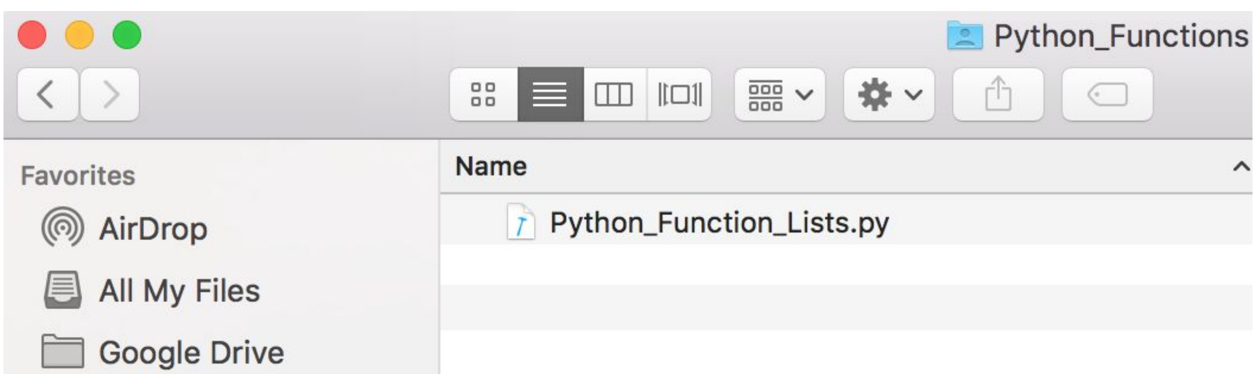
This section provides step-by-step instruction on how to use Python functions to access the hardware module on the Board.

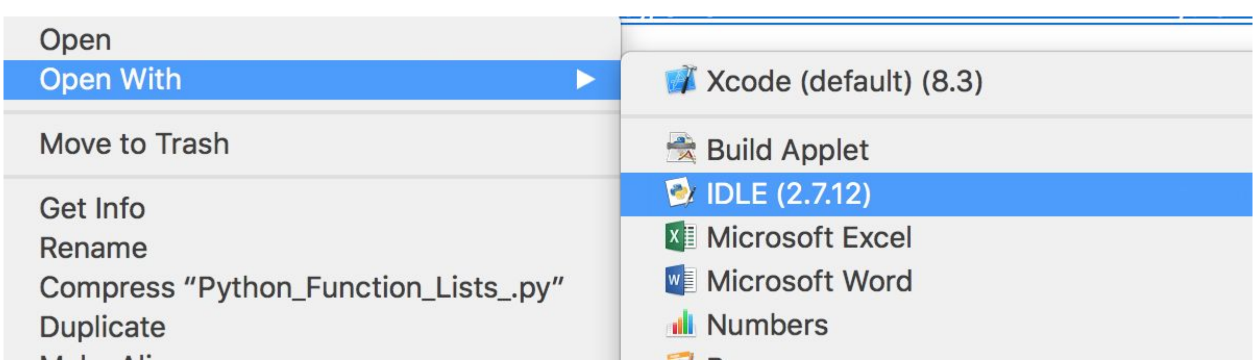
Note: The goal of this section is to help user to write their own automation python script by using those high level functions. Therefore, the demo here is to load those function and demonstrate the usages by calling them directly in the console (like you would do in your code).

The python function list is provided in the google doc folder.

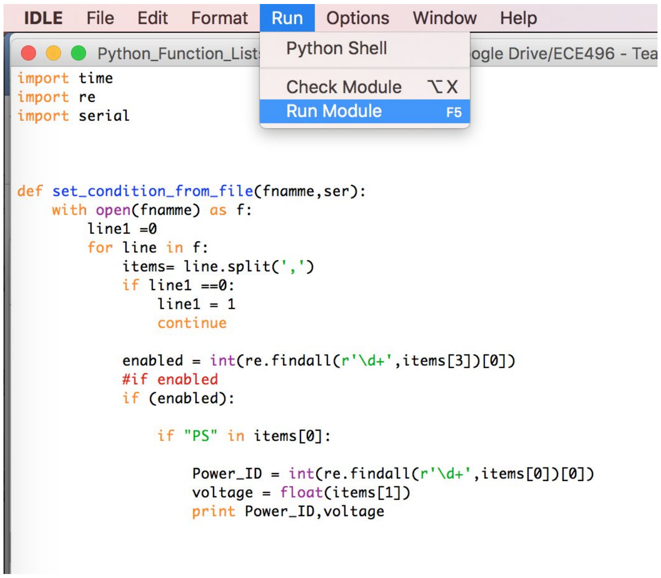
2.1 Setting Up the Python Function(Steps are same for both Mac and Windows user)

Step 1: Open up the python script with default python IDLE editor.





Step 2: Load the functions into the consoles



Loaded successfully!

Step 3:

Create a serial connection by calling

ser = connect(port_num)

The Port_num can be found:
#**Windows** : Open device manager etc COM4

#**MAC** : Open a terminal and Type "ls /dev/cu.*", It will look like ""/dev/cu.usbserial-A105RZJ1"

```
>>> ser = connect("/dev/cu.usbserial-A105RZJ1")
```

User will use this "ser" object for all later communications with the board.

2.2 Power Supply

Set_power(Power_supply_ID, voltage,ser):

#Power_supply_ID: Int, 1-7, 7= power supply for level shifter for GPIO

#voltage : 0.5-3 V

#ser : serial port object, obtained from called connect

Example: Setting power supply 3 to 1V

```
>>> Set_power(3, 1,ser):
```

Get_power_voltage(Power_supply_ID, ser):

#Power_supply_ID : 1-7, 7 = the power supply for level shifter for GPIO

#ser : serial port object, obtained from called connect

#return voltage

Example: Measure the output voltage at power supply 3

```
>>> PV3 = Get_power_voltage(3, ser)
```

Get_power_current(Power_supply_ID,ser):

#Power_supply_ID : 1-7, 7 = the power supply for level shifter for GPIO

#ser : serial port object, obtained from called connect

#return current_in_mA

Example: Measure the current at power supply 3

```
>>> CP3 = Get_power_current(3,ser)
```

2.3 Positive Bias Voltage

Set_pos_bias(Pos_Bias_ID,voltage,ser):

#Pos_Bais_ID: Int, 1-4

#voltage : 0 - 3V

#ser : serial port object, obtained from called connect

Example: Setting Positive Bias Voltage 1 to 1.8V

```
>>> Set_pos_bias(1,1.8,ser)
```

2.4 Negative Bias Voltage

Set_neg_bias(Pos_Bias_ID,voltage,ser):

#Pos_Bais_ID: Int, 1 or 2

#voltage : -3 - 0V

#ser : serial port object, obtained from called connect

Example: Setting Negative Bias Voltage 2 to -1.8V

```
>>> Set_neg_bias(2,-1.8,ser)
```

2.5 Bias Current

Set_current_bias(Current_Bias_ID, current,ser):

#Current_Bais_ID: Int, 1-4

#current : 0 - 1000, unit in uA

#ser : serial port object, obtained from called connect

Example: Setting Bias Current 3 to 500uA

```
>>> Set_current_bias(3,500,ser)
```

Get_Bias_current(Current_Bias_ID, current,ser):

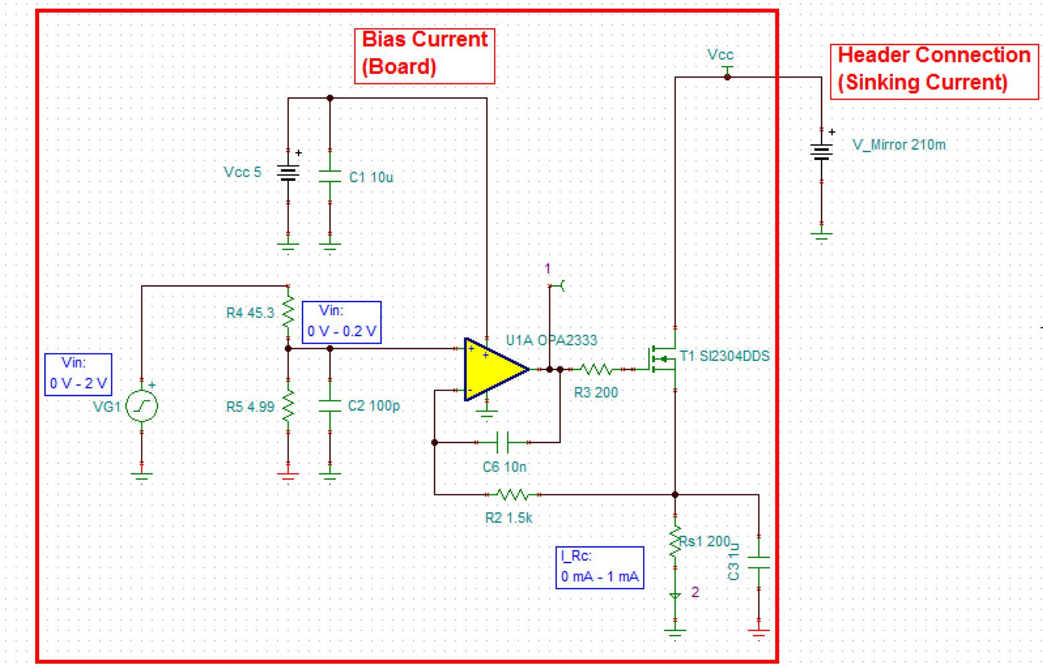
#Current_Bias_ID : 1-4,

#ser : serial port object, obtained from called connect

#return current in uA

Example: Measuring Bias Current 3 to 500uA

```
>>> BC3 = Get_Bias_current(3,ser)
```



Note : The Bias Current acts like a current-sink that sinks current from V_Mirror. The V_mirror needs to be larger than [i_bias * 200 Ohm] for the circuit to work properly.

For example, At maximum 1000uA, The minimum V_Mirror is 0.2V.

2.6 Variable Resistor

Set_resistance_2k5(Resistance,ser)

#Resistance : Resistance, 0-2500 , unit in Ohm

#ser : serial port object, obtained from called connect

Example: Setting Resistance to 1000

```
>>> Set_resistance_2k5(1000,ser)
```

Set_resistance_10k(Resistance,ser)

#Resistance : Resistance, 0-10000 , unit in Ohm

#ser: serial port object, obtained from called connect

Example: Setting Resistance to 5000

```
>>> Set_resistance_10k(5000,ser)
```

2.7 GPIO

set_frequency (frequency, ser)

#frequency: output data frequency in hz (1-20kHz)

#ser: serial port object, obtained from called connect

Example: Setting output data frequency to 20kHz

```
>>> set_frequency(20000,ser)
```

set_input_clock_cycles (cycles, ser)

#cycles: number of data in bits you want to capture on input (maximum 3900)

- captures both rising and falling edge data so the number of effective data for each pin is half of what is set. (ex. 1600 cycles is set, the number of rising edge data is effectively 800)

#ser: serial port object, obtained from called connect

Example: Setting input data cycles to 1600

```
>>> set_input_clock_cycles (1600, ser)
```

set_gpio (gpio_id, direction, clock_source, capture_mode, num_packets, data, ser)

#gpio_id: gpio identification (1-16)

- only pins 1-3 and 12-16 can be set as inputs

#direction: output (0) or input (1)

#clock_source: board clock (0) or DUT clock (1)

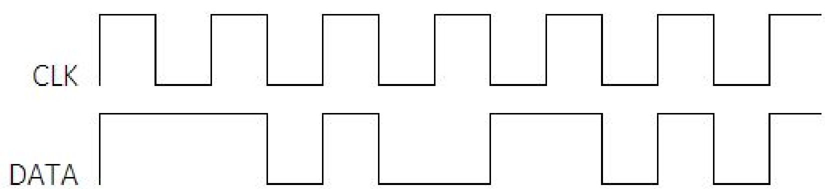
#**capture_mode**: rising (0), falling (1), input clock from DUT (2), asynchronous and uses board clock (3), output pin (4), output clock from board (5)

#**num_packets**: sets the number of data packets

- UART can only handle a maximum of 120 data bits per set_gpio command, if data exceeds 120 bits, num_packets need to be set to a number greater than 1; set num_packets = int (total data / 120) + 1.
- if there are more than 1 packet, use **load_extra_packets** function

#**data**: set output data in bits (ie. 101110101010...), maximum 120 bits

- if input pin or clock pin, set data to zero
- this data is defined for both clock high and low, for example, if the user wishes to have an output data such as the following, the output data bits would be set to 1110100110101



#**ser**: serial port object, obtained from called connect

Example: Set GPIO 1 as output clock

```
>>> set_gpio(1, 0, 0, 5, 1, 0,ser)
```

Example: Set GPIO 2 as input rising edge capture

```
>>> set_gpio(2, 1, 1, 0, 1, 0, ser)
```

Example: Set GPIO 3 as asynchronous capture using board clock

```
>>> set_gpio(3, 1, 0, 3, 1, 0, ser)
```

Example: Set GPIO 4 as output with data

```
>>> set_gpio(4, 0, 0, 4, 1, 1110101010111101010, ser)
```

Example: Set GPIO 12 as input clock

```
>>> set_gpio(12, 1, 1, 2, 1, 0, ser)
```

Example: Set GPIO 13 as input falling edge capture

```
>>> set_gpio(13, 1, 1, 1, 1, 0, ser)
```

load_extra_packets (data, ser)

#data: additional data for previously set gpio in bits, max 120 bits

#ser: serial port object, obtained from called connect

Example: set additional data

```
>>> load_extra_packets (1101010101110000, ser)
```

gpio_Begin (ser)

- run to begin data transmission, no inputs required

Example: start data transmission

```
>>> gpio_Begin(ser)
```

read_gpio_input (gpio_id, ser)

#gpio_id: input gpio identification that you wish to read from

Example: Read from GPIO 2

```
>>> read_gpio_input (2, ser)
```

Example: read data from function

```
>>> GD0211001111001100111110000
```

- The first 4 characters (GD02) are identifiers. "GD" is short for GPIO Data and "02" is the GPIO pin number
- Number read comes in pairs because data is read for both rising and falling edge. If this output was captured for rising edge, then the effective data is "10110101100"

Example of setting GPIOs, transmitting data, and reading back

```
set_frequency (10000, ser)          # sets board clock frequency to 10kHz
set_input_clock_cycles (10, ser)    # sets input capture to be 10 cycles

set_gpio (1, 1,0,3,1,0,ser)     # sets to input, asynchronous capture
set_gpio (2, 1,1,0,1,0,ser)     # sets to input, dut clock, rising capture
set_gpio (3, 1,1,1,1,0,ser)     # sets to input, dut clock, falling capture
set_gpio (4, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (5, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (6, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (7, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (8, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (9, 0,0,4,1,11101010101110101010,  ser)  # set to output with data
set_gpio (10,0,0,4,1,11101010101110101010, ser)  # set to output with data
set_gpio (11,0,0,5,1,0,  ser)   # sets to output clock source
set_gpio (12,1,1,2,1,0,  ser)   # sets to input clock source
set_gpio (13,1,0,3,1,0,  ser)   # sets to input, asynchronous capture
set_gpio (14,1,1,0,1,0,  ser)   # sets to input, dut clock, rising capture
set_gpio (15,1,1,1,1,0,  ser)   # sets to input, dut clock, falling capture
set_gpio (16,1,1,0,1,0,  ser)   # sets to input, dut clock, falling capture

gpio_Begin()                    # stats data transmission
read_gpio_input (1)             # read back data from gpio 1
```

Notes:

1. Prior to data transmission, output frequency and input clock cycles must be set.
2. Prior to beginning data transmission, gpio data for gpios 1-16 must be individually loaded and loaded in order.
3. If extra packets need to be loaded, it must be loaded immediately after gpio setting.
4. All output data length must be equal.
5. After data transmission, if user wishes to begin again, all gpio data must be reloaded in order regardless of whether or not the settings have changed.

2.8 SPI

The SPI module acts like SPI_Master to send file to the slave device.

SPI_SEND (data, Polarity, Phase, MSB, BitRate, ser)

#**data** : String, in Hex

#**Polarity** : 1 = Rising/Falling, 0 = Falling/Rising

#**Phase** : 1 = Sample/Setup, 0 = Setup/Sample

#**MSB** : 1 = Most Significant bit first, 0 = Least Significant bit first

#**Bitrate** : 0 - 500000

#**ser** : serial port object, obtained from called connect

Example: SPI sending data "9987654321" with

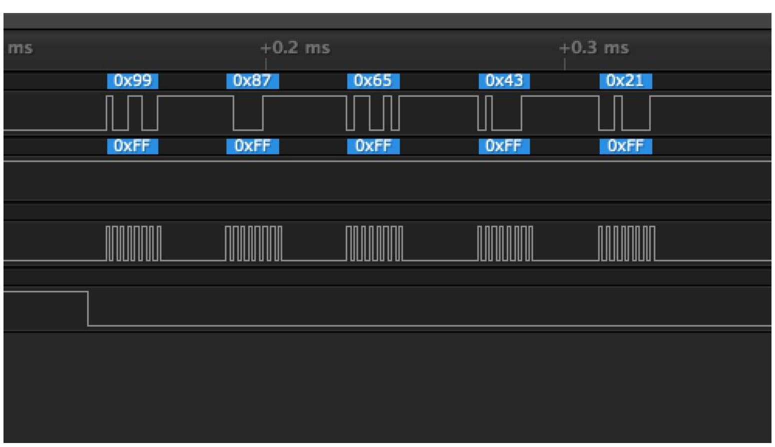
Polarity= Rising/Falling

Phase = Sample/Setup

MSB = 1

Bitrate = 100000

```
>>> SPI_SEND("9987654321",1,1,1,100000,ser)
```



SPI_disable ()

- Disables SPI mode to revert back to GPIO mode; no inputs required

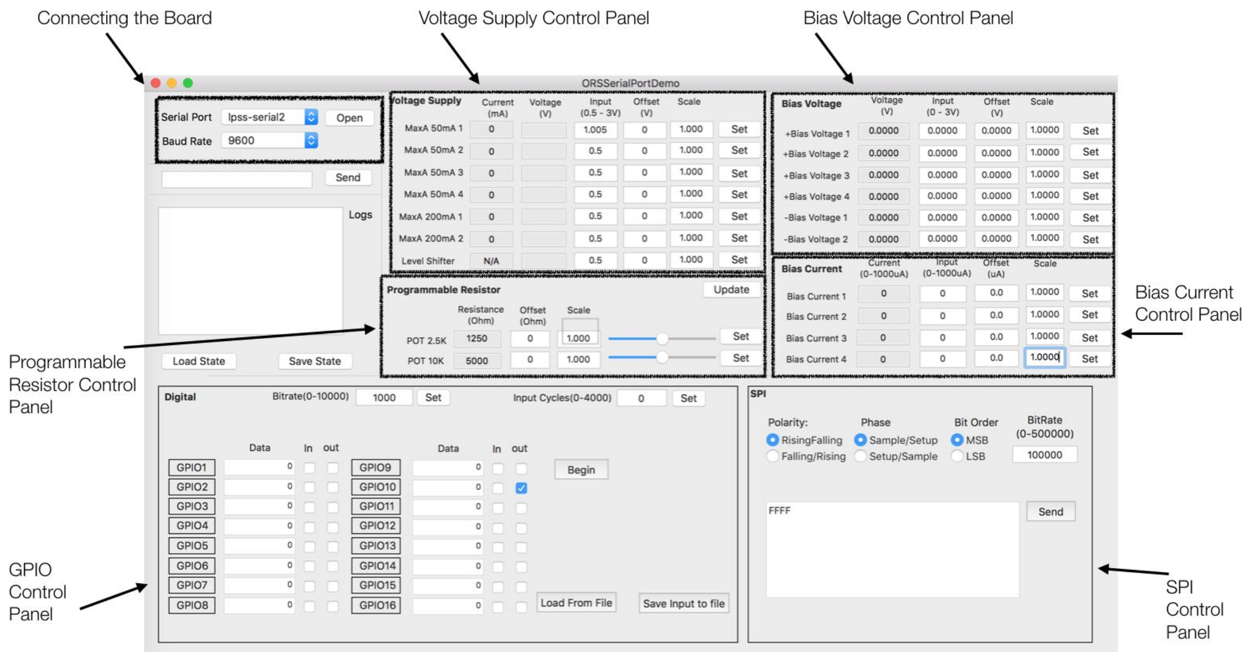
  Example: disable SPI

```
>>> SPI_disable (ser)
```

3.0 MAC OS App

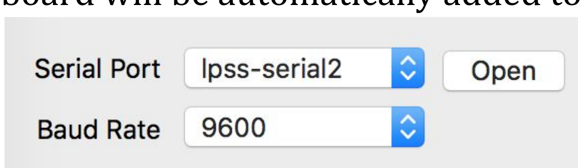
This section provides instruction on how to use the App in Mac OSto access the hardware module on the Board.

The APP can achieve the same functionality as the Python functions, It provides a simpler access to end user.



3.1 Connect to the board

Upon connecting the board to your Mac through USB, The serial port of the board will be automatically added to the drop box.



a. **Select the 9600 Baud Rate**
b. **Select the correct port . (Normally it will have a name "usbserial" in it)**
c. **Click open**

Now you have connected to the board

3.2 Voltage Supply Section

Voltage Supply	Current (mA)	Voltage (V)	Input (0.5 - 3V)	Offset (V)	Scale	
MaxA 50mA 1	0		1.005	0	1.000	Set
MaxA 50mA 2	0		0.5	0	1.000	Set
MaxA 50mA 3	0		0.5	0	1.000	Set
MaxA 50mA 4	0		0.5	0	1.000	Set
MaxA 200mA 1	0		0.5	0	1.000	Set
MaxA 200mA 2	0		0.5	0	1.000	Set
Level Shifter	N/A		0.5	0	1.000	Set

#**Current(mA)** : Current consumption,
#**Voltage(V)** : Real (Measured) the Voltage
#**Input(0.5 - 3V)** : Input Voltage
#**Offset(V)** : offset applying to the Input(0.5 - 3V)
#**Scale** : Scaling applying to the Input(0.5 - 3V)
#Set:

A. The program will take set the voltage to (input + Offset) * Scale
B. The program will take a Current measurement for this channel

Voltage supply are designed to have a Error of 10%,
However the, **Measured** voltage is accurate within +-3mV, so It allows user to accurately monitor the Output voltage.

The **offset** and **scale** allows user to perform manual errors corrections.
Voltage = (Input + Offset) * Scale

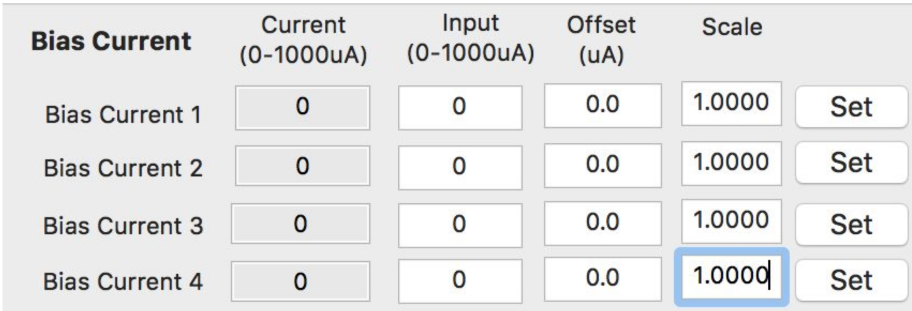
3.3 Bias Voltage

#**Voltage(V)** : Current Voltage (This is not measured Voltage)
#**Input(0 - 3V)** : Input Voltage
#**Offset(V)** : offset applying to the Input
#**Scale** : Scaling applying to the Input
#**Set:**

A. The program will take set the voltage to (input + Offset) * Scale
B. The program set the label Voltage(V) = (input + Offset) * Scale

The **offset** and **scale** allows user to perform manual errors corrections. If high accuracy is required
For example, If the DAC reference voltage have -0.01% error, This can be corrected with Scale factor 1.0001

3.4 Bias Current



#**Current(0 - 1000uA)** : Measured Current (This is not measured Voltage)
#**Input(0 - 1000uA)** : Input Current
#**Offset(uA)** : offset applying to the Input
#Set:

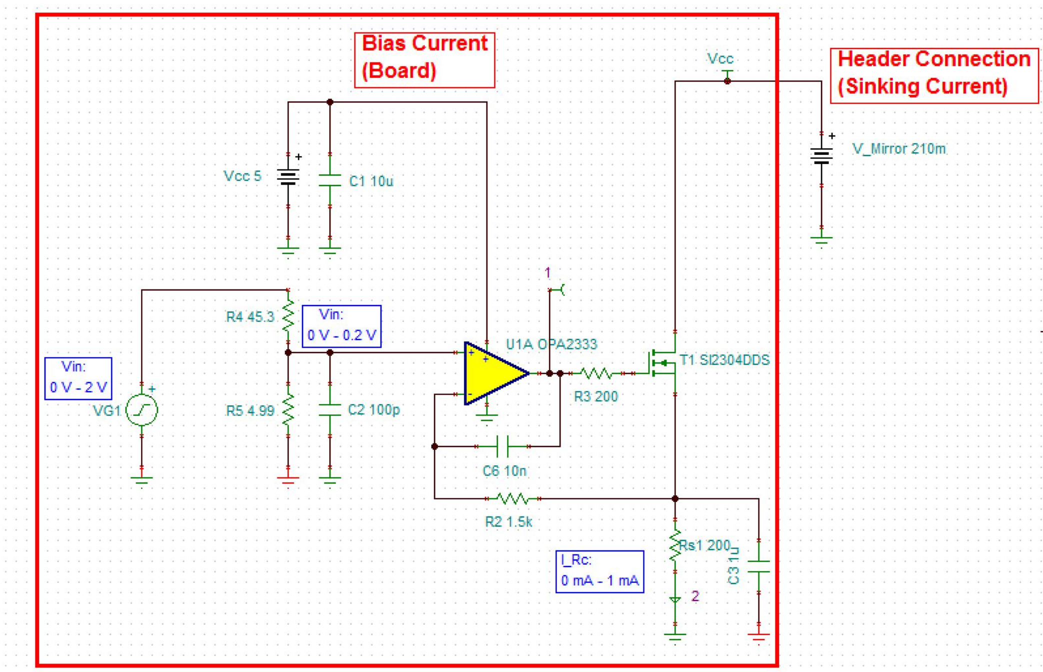
A. The program will set the Bias Current to (input + Offset) * Scale
B. The program will perform a current Measurement.

The **offset** and **scale** allows user to perform manual errors corrections. If high accuracy is required.

For example, If the resistor is rated with -0.01% error, This can be corrected with Scale factor 1.0001

Voltage = (Input + Offset) * Scale

Bias Current Circuit Configuration:



Note : The Bias Current acts like a current-sink that sinks current from V_Mirror. The V_mirror needs to be larger than [i_bias * 200 Ohm] for the circuit to work properly.

For example, At maximum 1000uA, The minimum V_Mirror needs to be 0.2V.

3.5 Programmable resistor

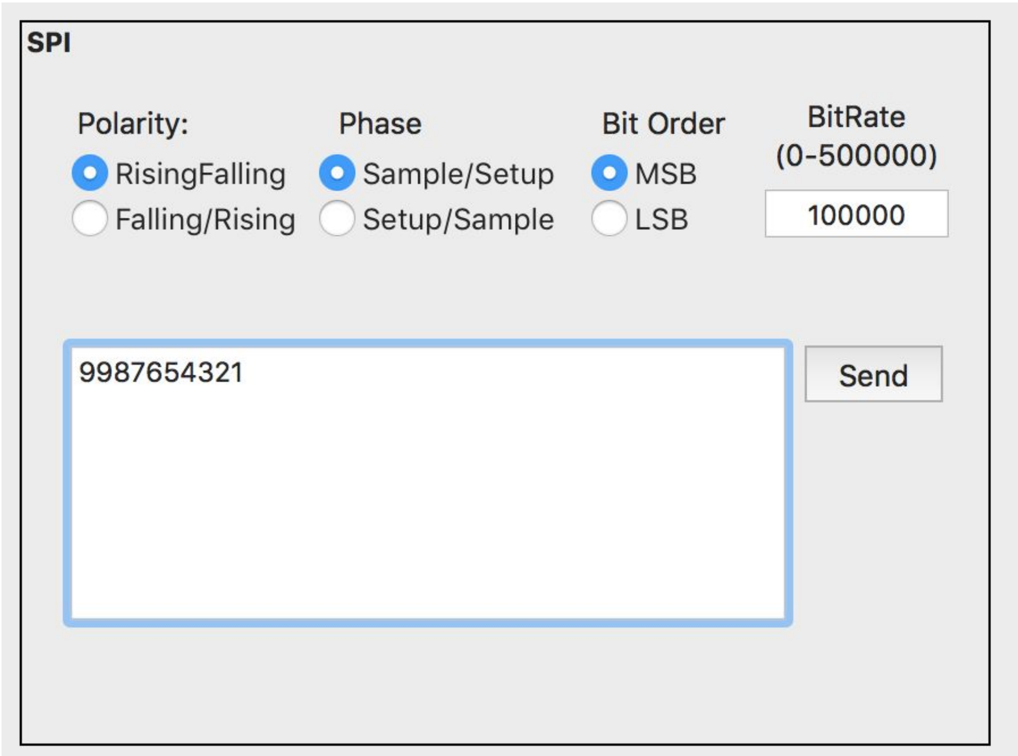
#**Resistance(Ohm)** : Resistance setted by user
#**Offset** : offset Resistance
#**Scale** : Scaling applying to the Input Resistance
#**Slidebar** : Input resistance, 0 Ohm = left_most, max = right_most
#**Set** :

A. Set Resistance to Resistance = (Input + Offset) * Scale
B. Update the Current Resistance Label.

The **offset** and **scale** allows user to perform manual errors corrections. If high accuracy is required

For example, If the resistor sits at 52 ohm at 0, Then a offset of -52 will result a more accurate resistance.

3.6 SPI

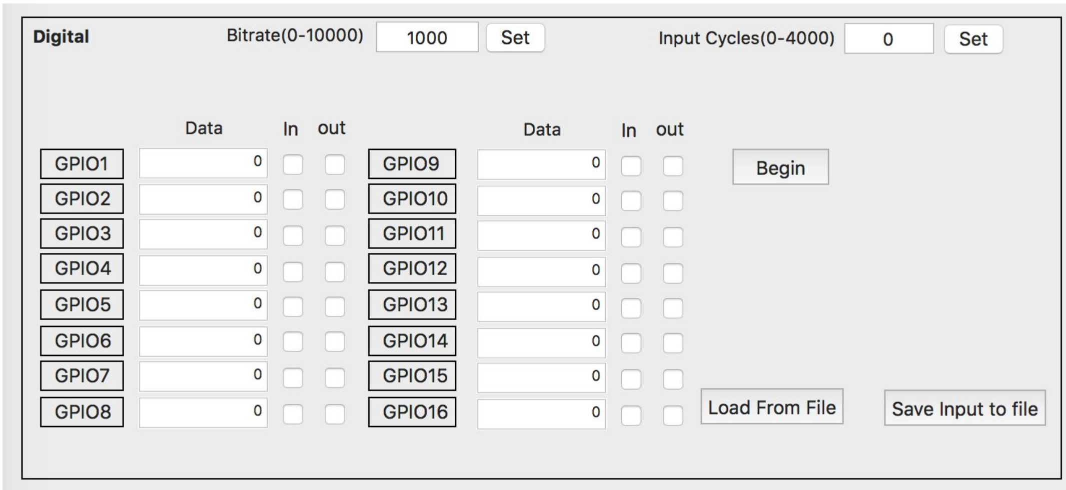


User will have the option to configure SPI and enter data in the Text box

#**Text_Box** : Message in Hex
#**BitRate**: 0 -500000 bps
#**Send**:

A. Configure the SPI, (Bit Rate, Polarity, Phase, Bit_Order)
B. Send the Data in the Text box.

3.7 Digital GPIO



3.7.1 User Interface

#**Load_From_File**:

A. Allow user to select an file that contains the setting and data for each GPIO (See Example on the file format below), up to 3900 bits of data

B. Load the setting and data from the file onto the Board.

C. Load first 10bits of the data onto the **Data** field and indicate the direction of each GPIO.

Begin :

A. Sets the bitrate and input cycle (0 if only outputting), up to 3900 input cycles

B. Start Outputting/Input the data

Save_Input_to_File :

A. Retrieve the data from board's input buffer and save to a file.

#**Data**: Contain first 10 bits of the output data string. (loaded from file, Not much use, only to make sure the program is working)

#**In/Out**: Indicating the direction of the GPIO.

3.7.2 Digital Configuration file format

Here is the format of the file for setting up the GPIO

	A	B	C	D	E
1		Direction(0=output;1	Clock_to_sync(0=board_c	Capture_Mode(0 = rising:	data(Binary)
2	GPIO1	1	0	3	0
3	GPIO2	1	1	0	0
4	GPIO3	1	1	1	0
5	GPIO4	0	0	4	b1110101010
6	GPIO5	0	0	4	b1010101010
7	GPIO6	0	0	4	b1111111111
8	GPIO7	0	0	4	b0000000000
9	GPIO8	0	0	4	b0011001100
10	GPIO9	0	0	4	b0111110000
11	GPIO10	0	0	4	b1001001001
12	GPIO11	0	0	5	0
13	GPIO12	1	1	2	0
14	GPIO13	1	0	3	0
15	GPIO14	1	1	0	0
16	GPIO15	1	1	1	0
17	GPIO16	1	1	1	0

#**Direction**:
- 0 : Output
- 1: Input

#**Clock_to_sync**:
- 0 : Board
- 1: DUT Clock

#**Capture Mode**:
- 0 : Rising
- 1: Falling
- 2: Input clock from DUT
- 3: Input is synced to board clock
- 4: Pin is an output
- 5: Output clock from Board

#**Data**:

Binary data, "b" added at beginning, **All the output pins need to have the save length of the data, if the output pins are not being used, set it to** 0

File Format:

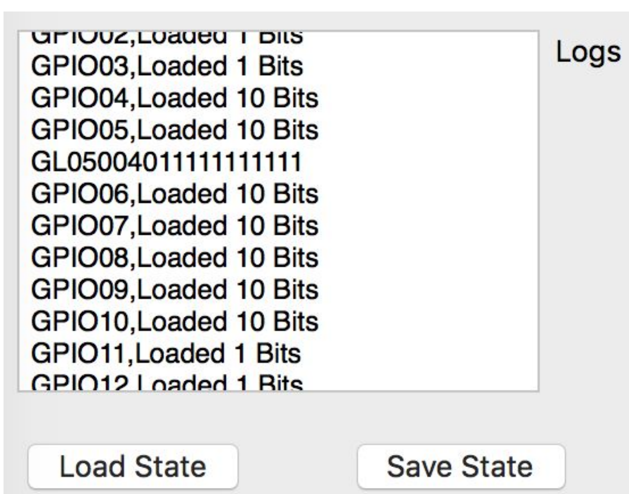
A. "," delimited between each column
B. First Row is expected to be label, Content does not matter
C. First Column is the ID of each GPIO, should not be changed.

```
,Direction(0=output;1=input),Clock_to_sync(0=board_clock;
GPIO1,1,0,3,0
GPIO2,1,1,0,0
GPIO3,1,1,1,0
GPIO4,0,0,4,b1110101010
GPIO5,0,0,4,b1010101010
GPIO6,0,0,4,b1111111111
GPIO7,0,0,4,b0000000000
GPIO8,0,0,4,b0011001100
GPIO9,0,0,4,b0111110000
GPIO10,0,0,4,b1001001001
GPIO11,0,0,5,0
GPIO12,1,1,2,0
GPIO13,1,0,3,0
GPIO14,1,1,0,0
GPIO15,1,1,1,0
GPIO16,1,1,1,0
```

Example On Outputting data with GPIO.

1. Connect to the board
2. Click **Load_From_File**
3. Select the example Digital Configuration file[Binary_Out_April_10th] provided in the folder [Digital_out_example]

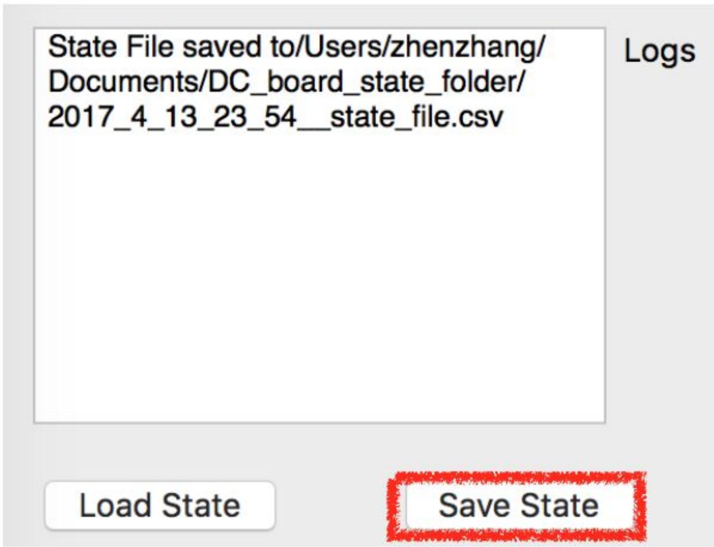
   The Log console will inform you the number of bits loaded onto the board



4. As the File set GPIO11 to be Clock, and GPIO4 - GPIO10 as outputs. The bit sequence can be captured on those pins.

	A	B	C	D	E
1		Direction(0=output;1	Clock_to_sync(0=boar	Capture_Mode(0 = rising:	data(Binary)
2	GPIO1	1	0	3	0
3	GPIO2	1	1	0	0
4	GPIO3	1	1	1	0
5	GPIO4	0	0	4	b1110101010
6	GPIO5	0	0	4	b1010101010
7	GPIO6	0	0	4	b1111111111
8	GPIO7	0	0	4	b0000000000
9	GPIO8	0	0	4	b0011001100
10	GPIO9	0	0	4	b0111110000
11	GPIO10	0	0	4	b1001001001
12	GPIO11	0	0	5	0
13	GPIO12	1	1	2	0
14	GPIO13	1	0	3	0
15	GPIO14	1	1	0	0
16	GPIO15	1	1	1	0
17	GPIO16	1	1	1	0

3.8. State Saving/Restoring Function



The Save State option allow user to save the state of the board into a file, and Restore the board to the same state on a later time. The state of board includes, state of Power supply voltage, Bias Voltage, Bias Current, and Programmable Resistors.

Upon clicking the button, The App will create a folder call "DC_board_state_folder". (if not existed already). And the file will be saved there and named in the format of "year_month_day_hour_minutes"

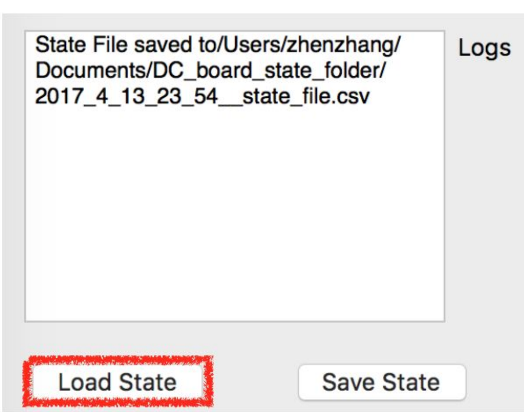
Here is the format of the state file, user can modify it in excel. The file is "," delimited, The name of first column should not be modified, and the the first row are expected to be labels.

#PS : Power Supply
#BV :Bias Voltage
#NBV: Negative Bias Voltage
#BC : Bias Current
#Enable: only update if Enable =1

	A	B	C	D	E
1		Load	Offset	Scaling	Enable
2	PS1	1.005	0	1	0
3	PS2	0.5	0	1	0
4	PS3	0.5	0	1	0
5	PS4	0.5	0	1	0
6	PS5	0.5	0	1	0
7	PS6	0.5	0	1	0
8	PS7	0.5	0	1	0
9	BV1	0	0	1.0001	0
10	BV2	0	0	1.0001	0
11	BV3	0	0	1.0001	0
12	BV4	0	0	1.0001	0
13	NBV1	0	0	1.0001	0
14	NBV2	0	0	1.0001	0
15	BC1	0	0	1.0001	0
16	BC2	0	0	1.0001	0
17	BC3	0	0	1.0001	0
18	BC4	0	0	1.0001	0
19	POT2K5	1250	0	1	0
20	POT10K	5000	0	1	0

```
,Load,Offset,Scaling,Enable
PS1,1.005,0,1.000,0
PS2,0.5,0,1.000,0
PS3,0.5,0,1.000,0
PS4,0.5,0,1.000,0
PS5,0.5,0,1.000,0
PS6,0.5,0,1.000,0
PS7,0.5,0,1.000,0
BV1,0.0000,0.0000,1.0001,0
BV2,0.0000,0.0000,1.0001,0
BV3,0.0000,0.0000,1.0001,0
BV4,0.0000,0.0000,1.0001,0
NBV1,0.0000,0.0000,1.0001,0
NBV2,0.0000,0.0000,1.0001,0
BC1,0,0.0,1.0001,0
BC2,0,0.0,1.0001,0
BC3,0,0.0,1.0001,0
BC4,0,0.0,1.0001,0
POT2K5,1250,0,1.000,0
POT10K,5000,0,1.000,0
```

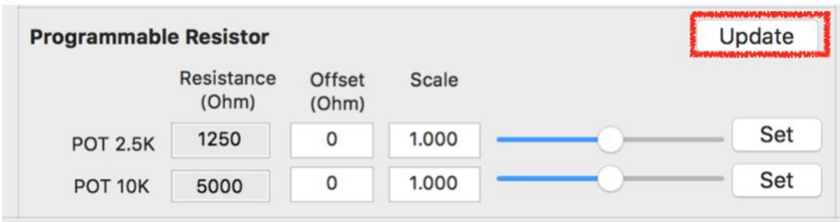
User can click the Load state button and select the state file to restore to the previous state. (**This will automatic set the board state**)



3.9. Update Function

Upon Clicked

A. It will check which module is currently being used
B. Measured the current and voltage of those modules
C. update the values on the user interface.



Appendix:

Validation Results

1. **Positive Bias Voltage**

Bias Voltage_Channel1				
Expected(V)	DAC CODE	DAC_HEX	Measurements(V)	Error(mV)
0.01	218.4533333	DA	0.009883	0.000117
0.02	436.9066667	1B5	0.01991	0.00009
0.05	1092.266667	444	0.049883	0.000117
0.1	2184.533333	889	0.099893	0.000107
0.5	10922.66667	2AAB	0.49983	0.00017
1	21845.33333	5555	0.9974	0.0026
1.5	32768	8000	1.4996	0.0004
2	43690.66667	AAAB	1.9995	0.0005
2.8			2.7994	0.0006

Bias Voltage_Channel2				
Expected(V)	DAC CODE	DAC_HEX	Measurements(V)	Error(mV)
0.01	218.4533333	DA	0.0100022	-0.0000022
0.02	436.9066667	1B5	0.020046	-0.000046
0.05	1092.266667	444	0.050016	-0.000016
0.1	2184.533333	889	0.10002	-0.00002
0.5	10922.66667	2AAB	0.49992	0.00008
1	21845.33333	5555	0.99976	0.00024
1.5	32768	8000	1.4996	0.0004
2	43690.66667	AAAB	1.9995	0.0005
2.8			2.7993	0.0007

Bias Voltage_Channel3

Expected(V)	DAC CODE	DAC_HEX	Measurements(V	Error(mV)
0.01	218.4533333	DA	0.009974	0.000026
0.02	436.9066667	1B5	0.019998	0.000002
0.05	1092.266667	444	0.049966	0.000034
0.1	2184.533333	889	0.099971	0.000029
0.5	10922.66667	2AAB	0.49988	0.00012
1	21845.33333	5555	0.99973	0.00027
1.5	32768	8000	1.4997	0.0003
2	43690.66667	AAAB	1.9996	0.0004
2.8			2.7993	0.0007

Bias Voltage_Channel4

Expected(V)	DAC CODE	DAC_HEX	Measurements(V	Error(mV)
0.01	218.4533333	DA	0.010072	-0.000072
0.02	436.9066667	1B5	0.020092	-0.000092
0.05	1092.266667	444	0.050061	-0.000061
0.1	2184.533333	889	0.100067	-0.000067
0.5	10922.66667	2AAB	0.49995	0.00005
1	21845.33333	5555	0.99985	0.00015
1.5	32768	8000	1.4996	0.0004
2	43690.66667	AAAB	1.9995	0.0005

2. Negative Bias Voltage

Negtive_Bias Voltage_Channel2					
Expected(V)	DAC CODE	DAC_HEX	Measurements-(V)	Error(mV)	Error_%
-0.01	-218.4533333	DA	-0.010144	0.0001	-1.42%
-0.02	-436.9066667	1B5	-0.020154	0.0002	-0.76%
-0.05	-1092.266667	444	-0.050095	0.0001	-0.19%
-0.1	-2184.533333	889	-0.100073	0.0001	-0.07%
-0.5	-10922.66667	2AAB	-0.4998	-0.0002	0.04%
-1	-21845.33333	5555	-0.99943	-0.0006	0.06%
-1.5	-32768	8000	-1.4991	-0.0009	0.06%
-2	-43690.66667	AAAB	-1.9987	-0.0013	0.07%

3.Bias Current

Bias Current_Channel1		
Expected(uA)	Measurements(uA)	%_Error
10	9.9	1.01%
50	49.7	0.60%
200	199.3	0.35%
500	498.5	0.30%
600	598.3	0.28%
900	897.9	0.23%

Bias Current_Channel2		
Expected(uA)	Measurements(uA	%_Error
10	9.9	1.01%
50	49.8	0.40%
200	200.1	-0.05%
500	500.7	-0.14%
600	601	-0.17%
800	801.3	-0.16%

Bias Current_Channel3		
Expected(uA)	Measurements(uA	%_Error
10	10.05	-0.50%
50	50.03	-0.06%
200	200	0.00%
500	499.8	0.04%
600	599.45	0.09%
900	899.45	0.06%

Bias Current_Channel4		
Expected(uA)	Measurements(uA	%_Error
10	9.3	7.53%
50	49.1	1.83%
200	199	0.50%
500	498.7	0.26%
600	598.7	0.22%
900	898.5	0.17%

4. Voltage Supply

Power Supply_1			
@~30mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7419	0.74218	-0.28
0.9V	0.9031	0.90281	0.29
1.2V	1.2301	1.22985	0.25
1.5V	1.5446	1.54522	-0.62
1.8V	1.8665	1.86717	-0.67

Power Supply_2			
@~30mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7212	0.72054	0.66
0.9V	0.8747	0.87452	0.18
1.2V	1.1663	1.16512	1.18
1.5V	1.4511	1.45214	-1.04
1.8V	1.7386	1.73721	1.39

Power Supply_3			
@~30mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7625	0.7616	0.90
0.9V	0.9526	0.9532	-0.60
1.2V	1.2751	1.276	-0.90
1.5V	1.5833	1.5821	1.20
1.8V	1.8837	1.882	1.70

Power Supply_4			
@~30mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7834	0.7824	1.00
0.9V	0.9766	0.9753	1.30
1.2V	1.2634	1.2622	1.20
1.5V	1.6093	1.6097	-0.40
1.8V	1.9564	1.9553	1.10

Power Supply_5			
@~150mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7671	0.768	-0.90
0.9V	0.9623	0.9631	-0.80
1.2V	1.3856	1.3839	1.70
1.5V	1.6984	1.6972	0.30
1.8V	1.9232	1.9214	1.80

Power Supply_6			
@~150mA	Multimeter_Measured(V)	ADC_Measured(V)	ERROR(mV)
0.742V	0.7923	0.7915	0.80
0.9V	1.0285	1.028	0.50
1.2V	1.449	1.4505	-1.50
1.5V	1.7251	1.7241	1.00
1.8V	1.9843	1.9835	0.80

5. **SPI**

```
>>> SPI_SEND("9987654321",1,1,1,100000,ser)
```