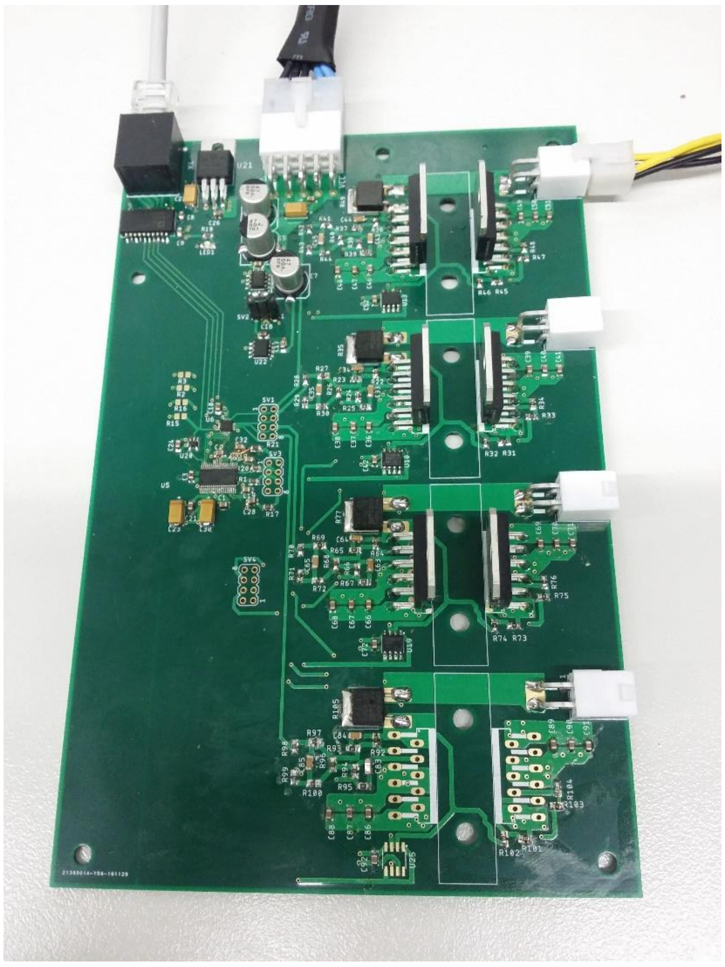
Manual for "GPA Dortmund" Version 1.1

General Overview

The design of this Gradient Power Amplifier (GPA Dortmund) is based on a GPA designed by Jason Stockmann and others. Changes that have been made include:

- combining the single SPI interfaces into one SPI interface
- adding circuitry for current sensing
- automatic calibration feature
- uses only single supply
- adding status output pins to check for overtempt of power OP amps
- 4 gradient channels



Function Overview (for one channel)

The 16-bit DAC80504 DAC controls a differential push-pull Howland current source. The Howland current source uses a 200 milliohms shunt resistor, which is also used to measure the current. The voltage across the shunt resistor is amplified by an INA149 OP amp, which is connected to a 2.5V offset voltage, to enable measuring bidirectional currents. The output of the INA149 is connected to an ADS8684 DAC.

The calibration is performed by setting the DAC to evenly distributed values over the whole range. The output current is then measured for each of these values. To output a user specific current, first order interpolation from the recorded values is done, to obtain the DAC value for the specific current.

Connecting the PCB to the power supply

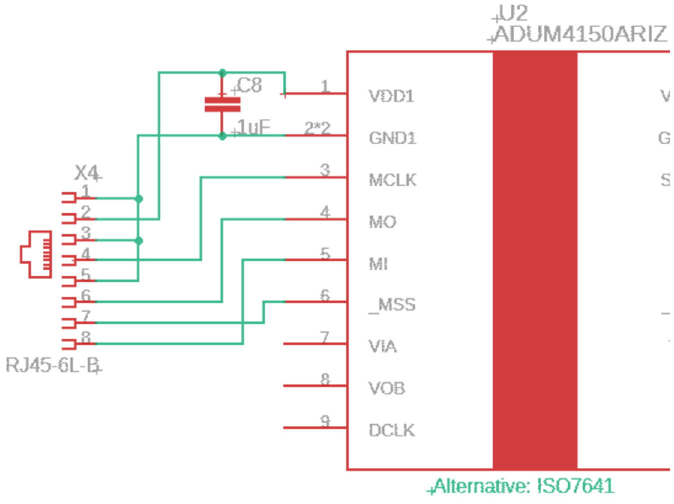
There are onboard voltage regulators for +12V and +5V. The only devices that use the input supply voltage (VCC) directly are the OPA549 power op amps. They have single supply maximum rating of 60 V.

For most applications a VCC of 15 V is enough. However, if faster rise times are necessary or if the coils have a very high resistance, higher voltages might be necessary. **It should be noted that a higher supply voltage causes higher heat losses in the power OP amps**.

Connecting the PCB to the via SPI to the microcontroller

The GPA has a galvanically isolated SPI interface using the ISO7641 from Analog Devices to prevent any ground loops. The ISO7641 has separate power pins for both sides and can operate from 2.7, 3.3 and 5 V logic levels. The maximum supply voltage is 6 V. The maximum SPI speed is 150 Mbps, however the DAC80504 supports a maximum SPI clock of 50 MHz and the ADS8684 supports a maximum SPI clock of 16 MHz.

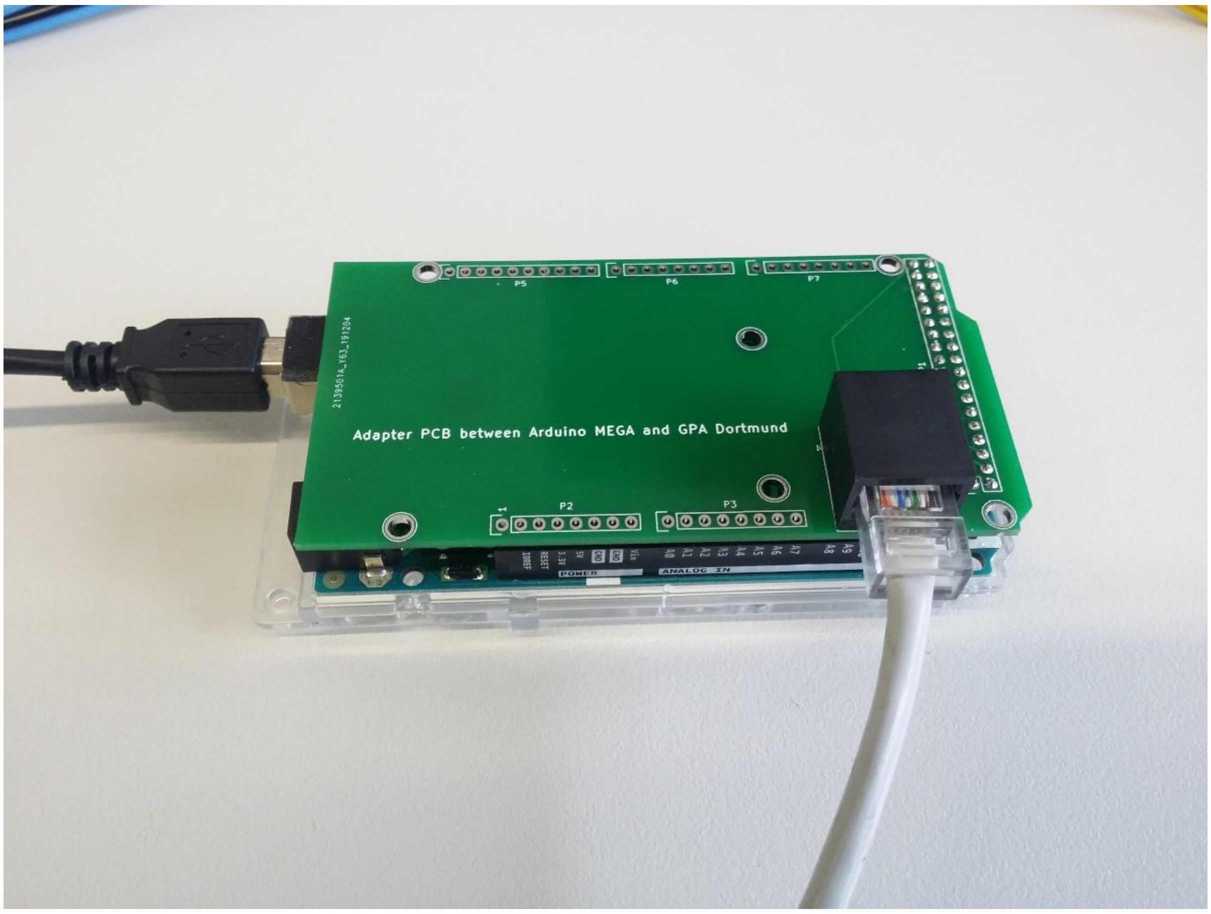
The connector for the SPI is a standard RJ45 connector. The pinout is:



Pin	Color	Function
1	white/orange	GND
2	orange	VDD
3	white/green	GND
4	blue	MCLK
5	white/blue	GND
6	green	MO
7	white/brown	_MSS
8	brown	MI

The abbreviations are: MI (master in), MO (master out), MCLK (master clock), _MSS (chip select).
_MSS activates the DAC, _MSS low activates the ADC.

We have designed an Arduino shield to allow connecting the Arduino to the GPA via a standard network cable.



Connecting the PCB to the gradient coils (or dummy loads)
The coils or dummy loads should be connected to the GPA with at least AWG 16 wire.

Controlling the DAC and ADC via SPI

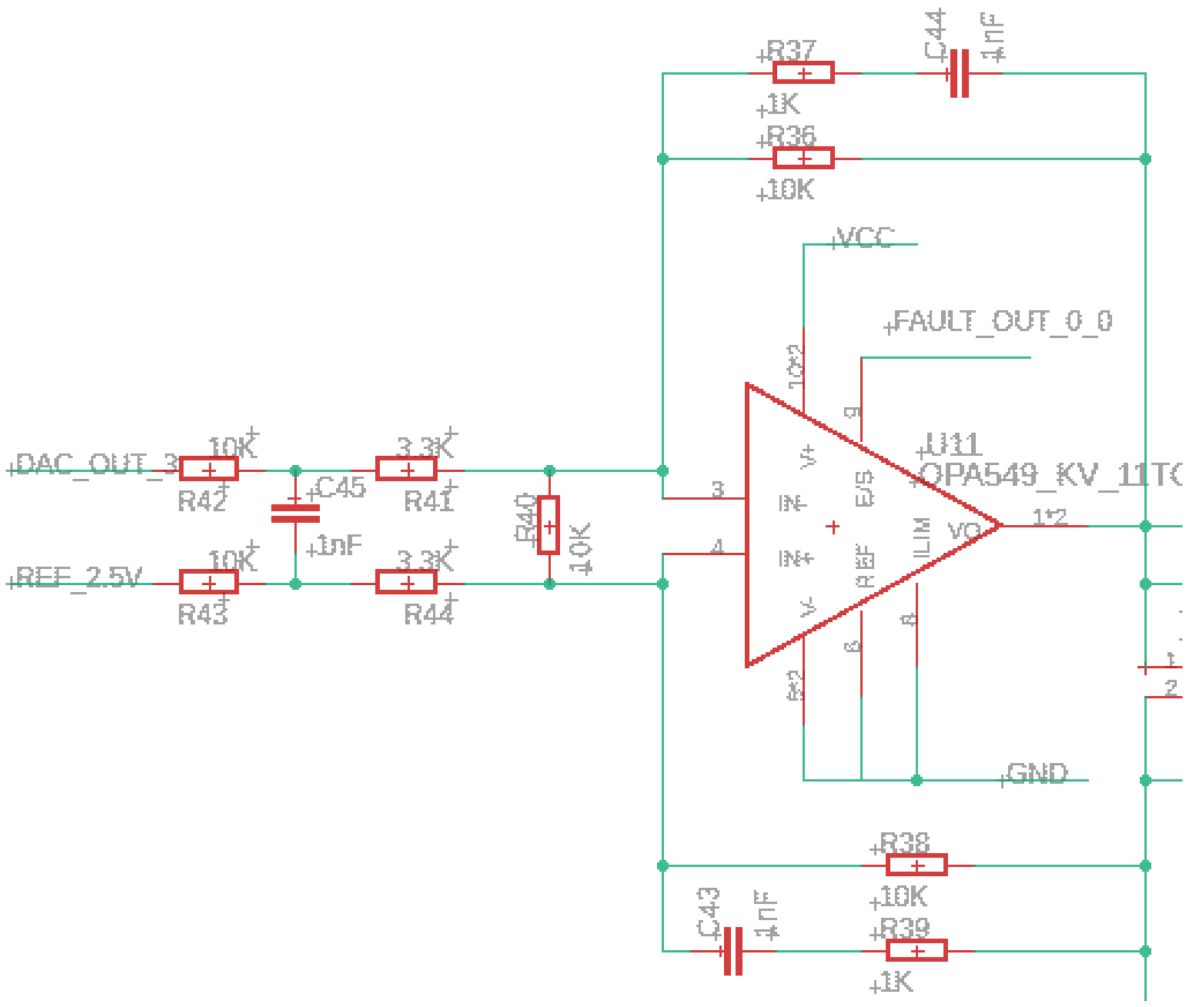
Arduino sample code in C++ is provided, which can do simple tasks like performing a calibration, doing a ramp for linearity measurements or setting the output current to a specific value.

We have tested the code on an Arduino MEGA. The program is controlled via Terminal with 115200 baud. The commands are as follows:

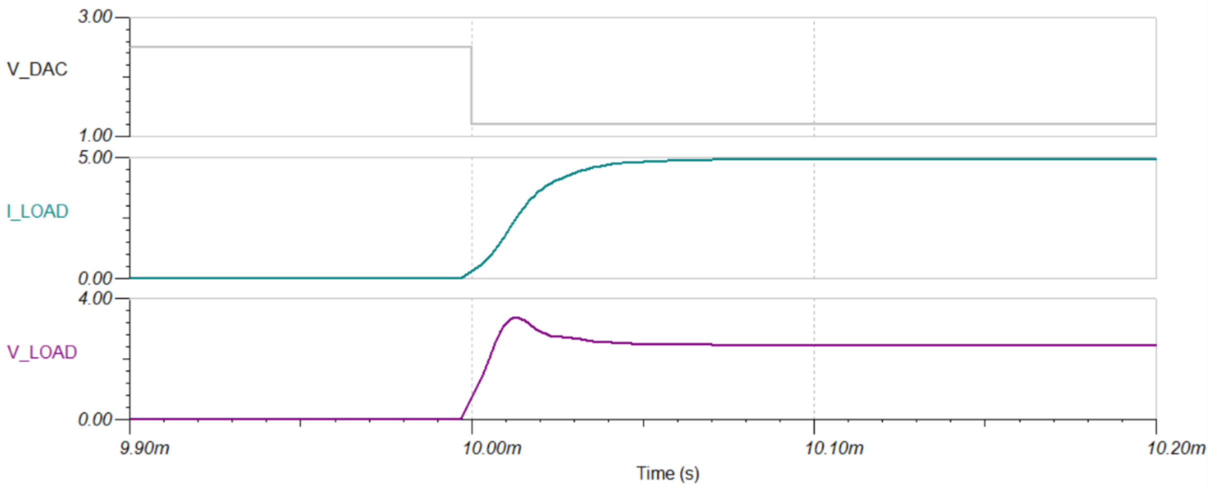
- f: searches the DAC value that corresponds to 0 A current. Always run this command before doing a calibration.
- g: perform calibration on gain factors.
- r: run a ramp. The ramp min (rampMinAmp) and max (rampMaxAmp) current need to be set in the source code. If the ramp is going over high currents (i.e. > 3A), a duty cycle can be switched on, to prevent OP amp overheating. The sleep time after each output can be defined in the source code (rampCyclePause).
- i: sets output current to 0 A and goes into idle mode.
- q: outputs configuration on terminal.
- p: outputs current pulses for rise time measurements. The high (pulseHighAmp) and low (pulseHighAmp) value of the pulse can be defined in the source code. The on time is 1/100 of a 100 ms cycle.
- cx: changes active channel to ‘x’. x can be a number between 1 and 4. Example “c4” sets active channel to 4.
- sx: sets the output current of the active channel to x. x can be only a single digit signed integer number. Example “s-2” sets the active channel current to -2 A.

Choosing the right filter-resistor values

The push power op amp is connected in the following fashion:



Capacitor values C44 and C43 need to be adjusted to the inductance of the gradient coils used. For the test setup we used a load with 10 uH inductance and 500 milliohms resistance. The SPICE simulation indicated that a capacitor value of 1 nF is stable with enough margin. The simulated rise-time was about 35 us.



Measured Performance

Rise-Time: 32 microseconds
Linearity (+-2A): INL 12 LSB which equals about 4.5 mA (= 0.075%)
Linearity (+-6A): INL 30 LSB which equals about 11.5 mA (= 0.192%)
Ripple: none

Conditions:
Coils used were the top and bottom x-gradients of the tabletop MRI plus their corresponding shields. The coils can be found here.
The total inductance of this coils were 10 uH, the resistance was 50 milliohms.

The supply voltage was 15 V, the power supply used was a Rohde and Schwarz HMP4040.

A current step from 0 to 2 A was applied via an appropriate SPI command to the DAC.

The Rise-Time and Ripple was measured across the shunt-resistor on the PCB.

Linearity was evaluated by reading current values from the ADC via SPI while applying a ramp on the DAC (see ramp function in the supplied Arduino sample code).

Calibration was performed over 33 points using first order lagrangian polynomials (linear interpolation). Using second order lagrangian polynomials would probably further improve linearity.

Maximum Values

Value	Minimum	Maximum
+VCC	12 V	60 V
Single Gradient Current		10 A
Total Current		40 A
SPI supply voltage	2.7 V	6 V
SPI clock frequency	100 kHz	16 MHz

License

open source hardware

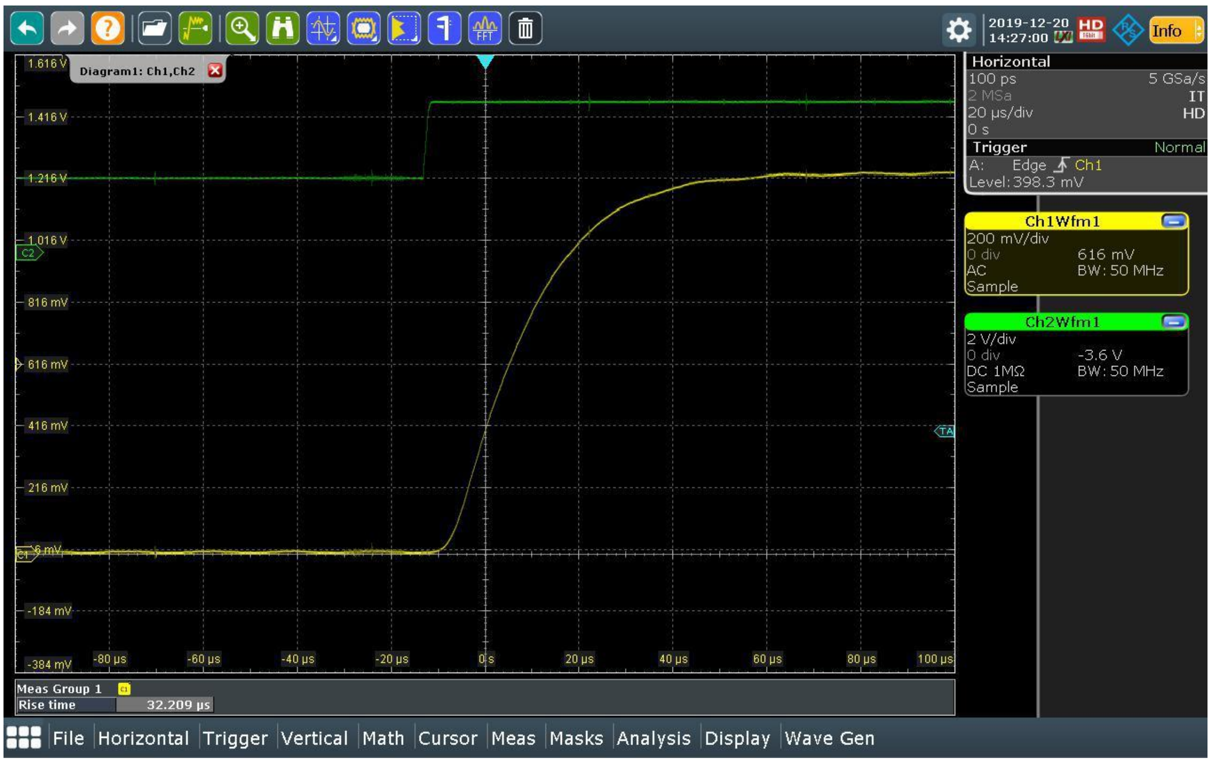
open source, open hardware.

Contact

Prof. Dr. Benjamin Menkuec
benjamin.menkuec@fh-dortmund.de

Appendix

Rise time measurement



Horizontal
100 ps 5 GSa/s
2 MSa IT
20 µs/div HD
0 s

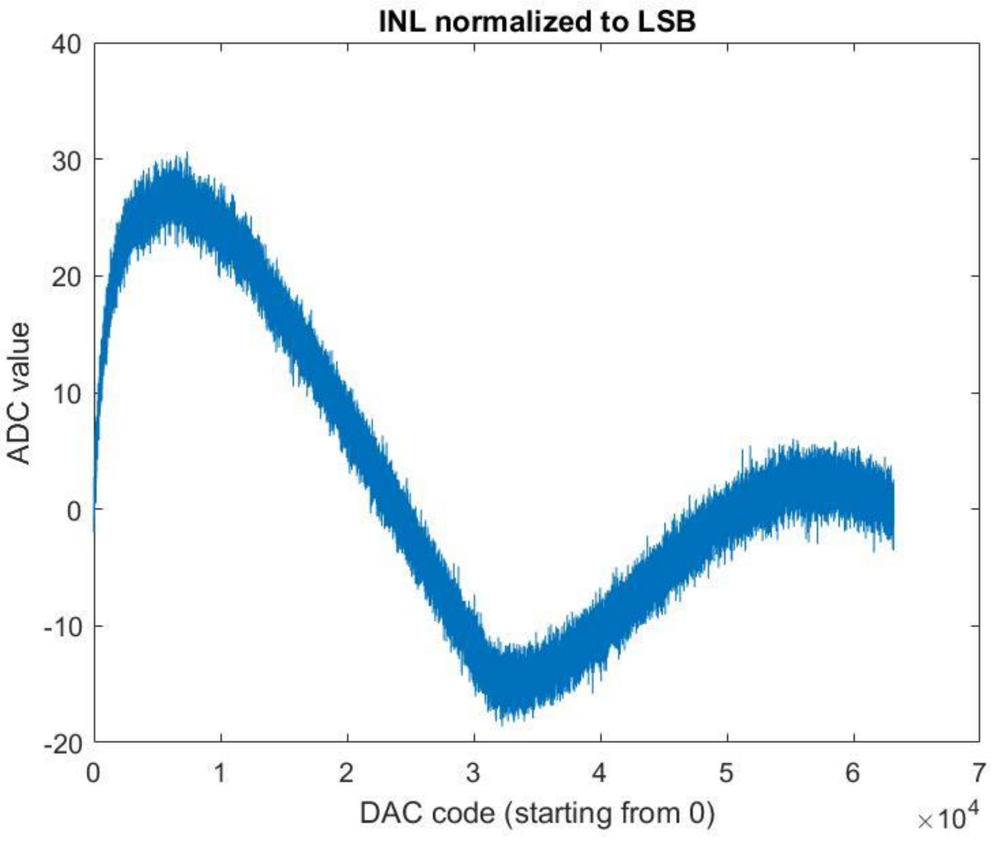
Trigger Normal
A: Edge Ch1
Level: 398.3 mV

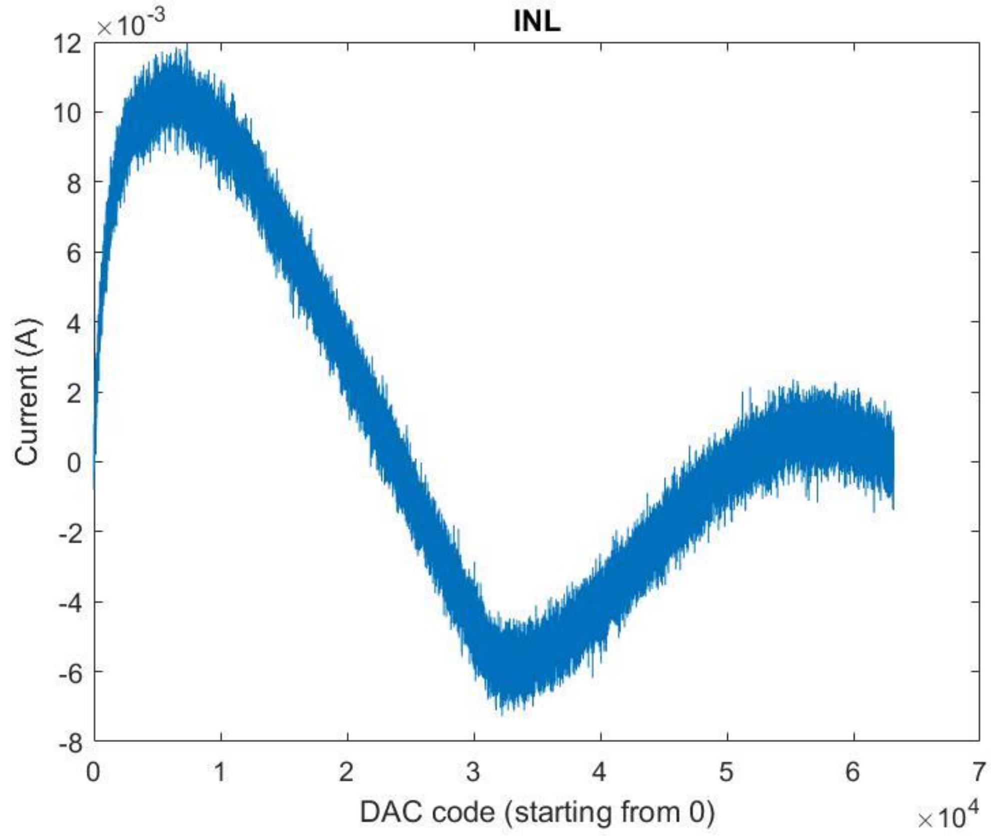
Ch1Wfm1
200 mV/div
0 div 616 mV
AC BW: 50 MHz
Sample

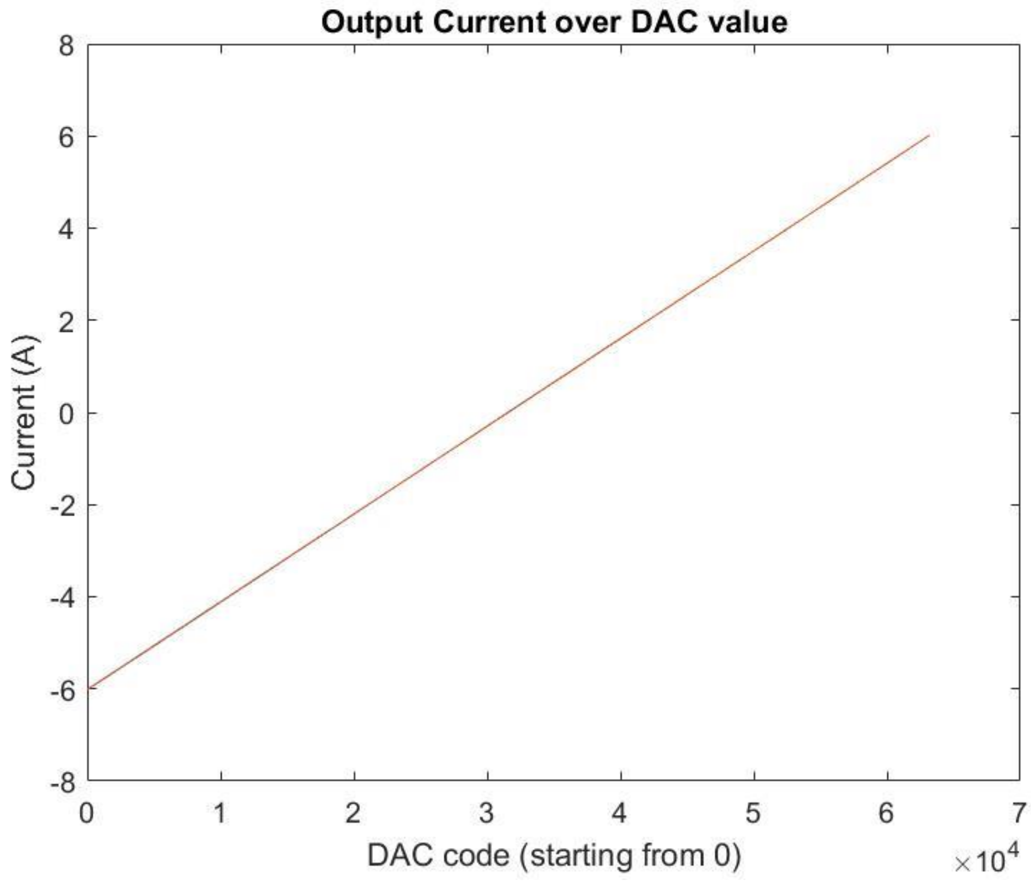
Ch2Wfm1
2 V/div
0 div -3.6 V
DC 1MΩ BW: 50 MHz
Sample

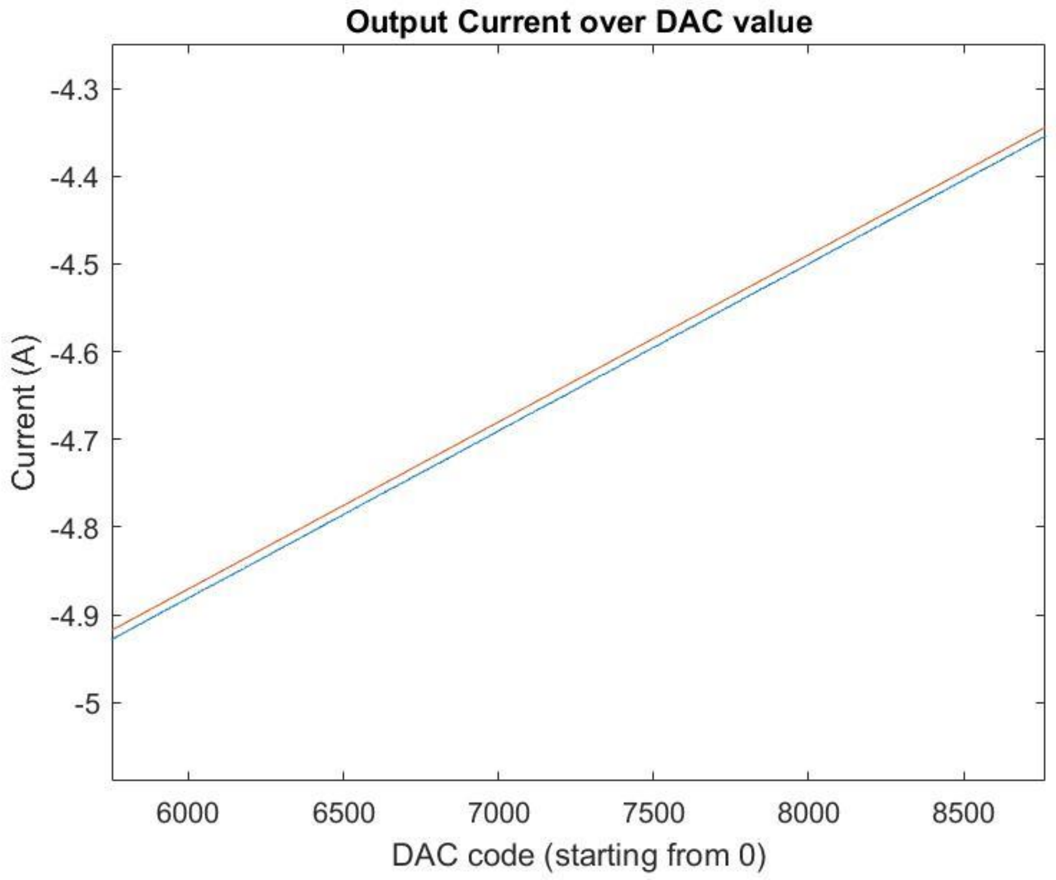
Meas Group 1
Rise time 32.209 µs

Linearity measurement for +-6 A:

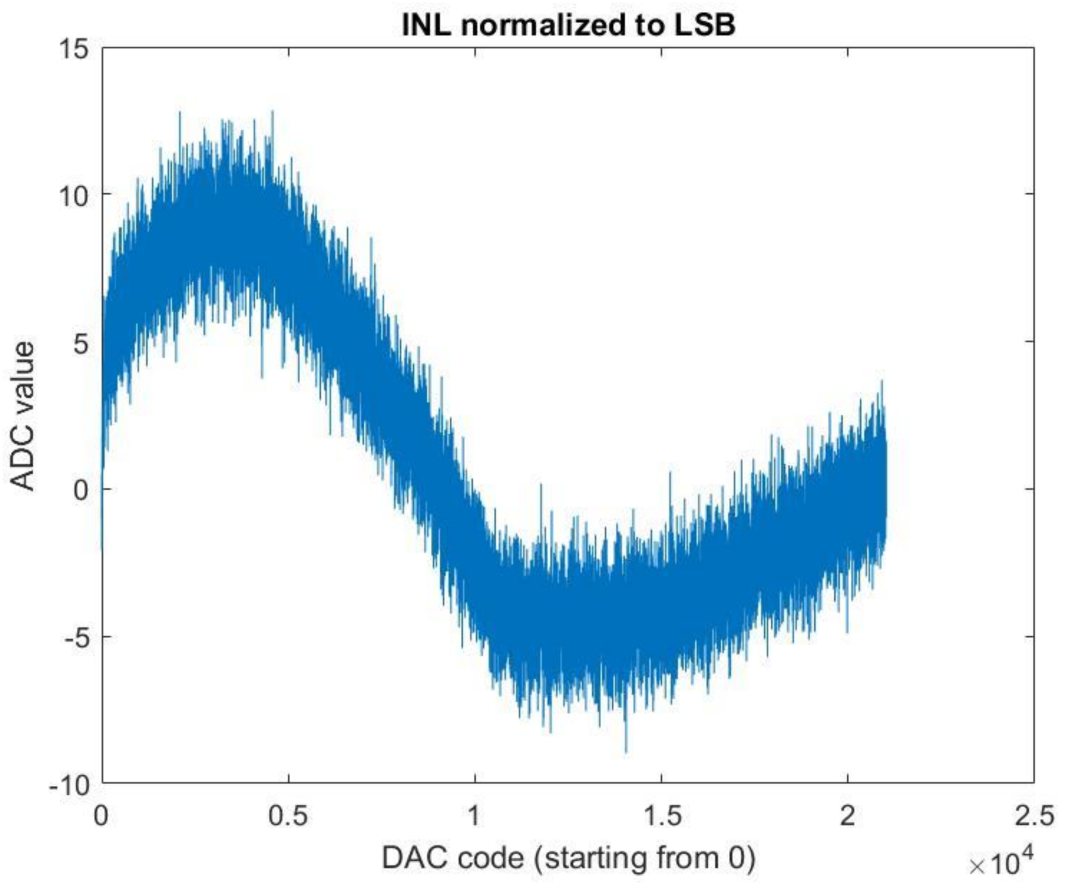


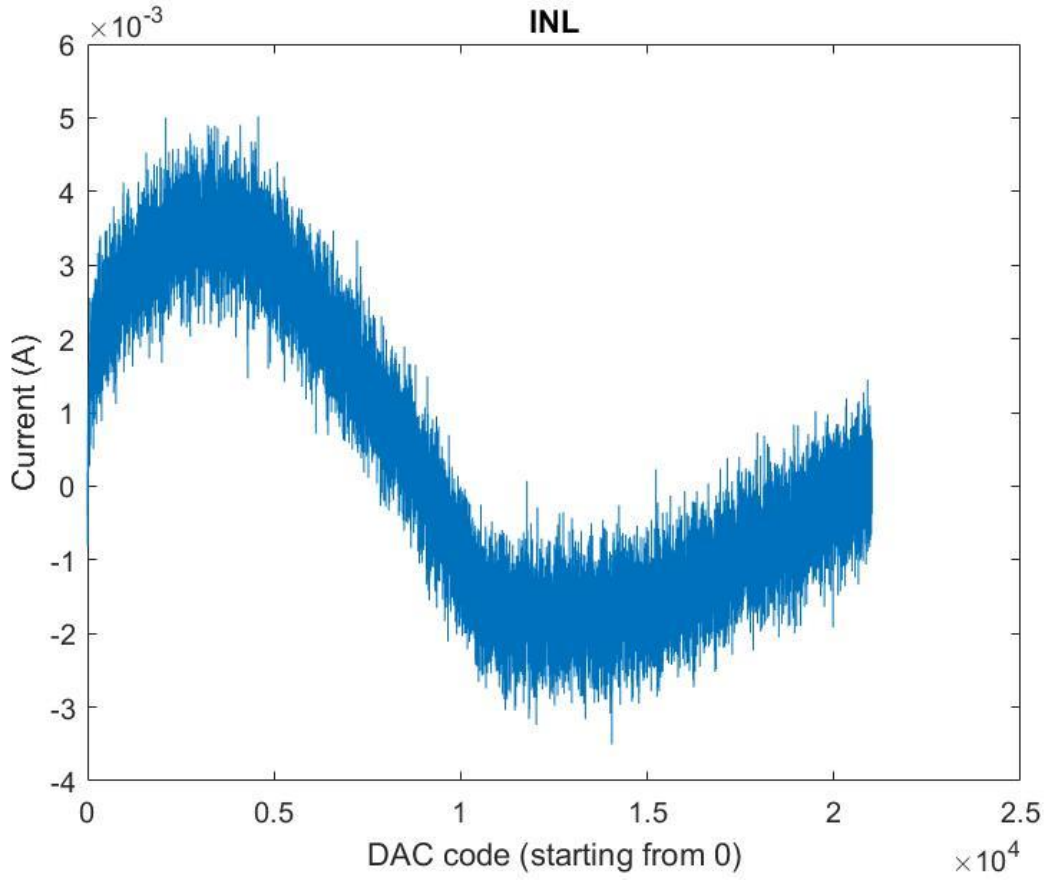



Output Current over DAC value
Current (A)
DAC code (starting from 0)
×10^4
8
6
4
2
0
-2
-4
-6
-8
0
1
2
3
4
5
6
7

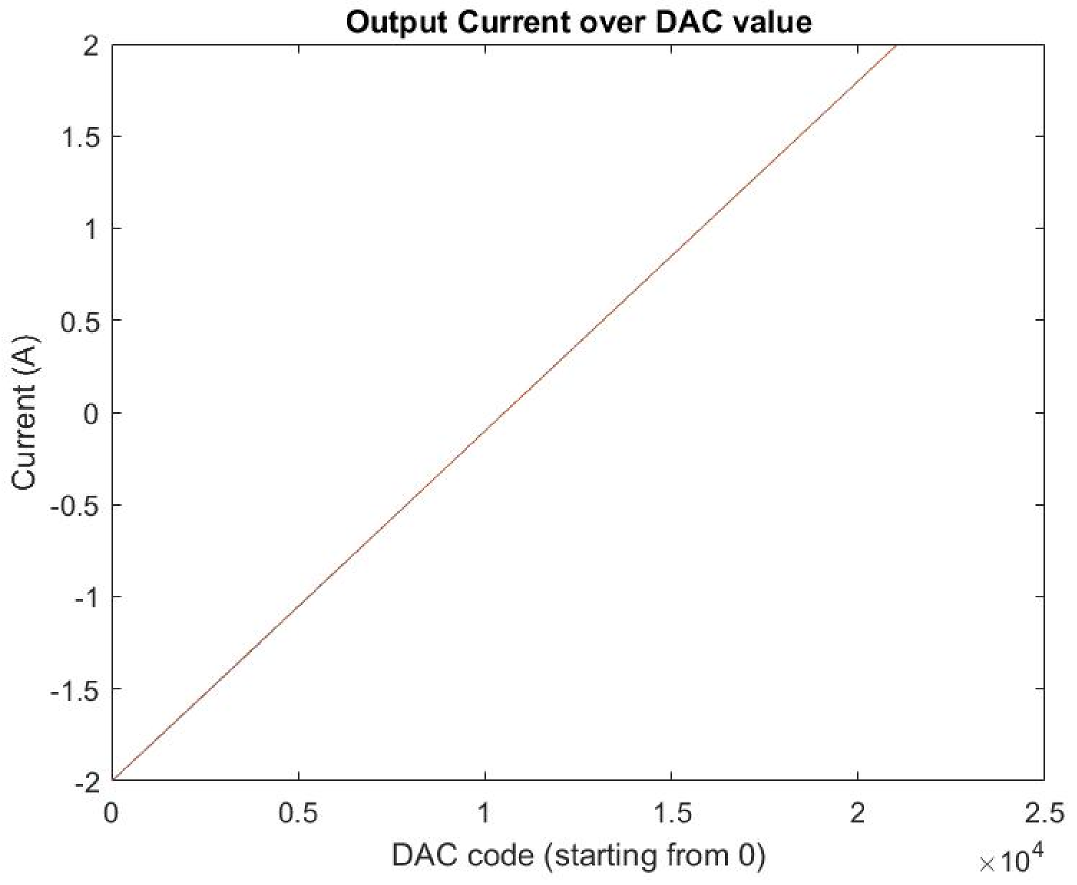

Output Current over DAC value
Current (A)
DAC code (starting from 0)
-4.3
-4.4
-4.5
-4.6
-4.7
-4.8
-4.9
-5
6000
6500
7000
7500
8000
8500

Linearity measurement for +-2 A

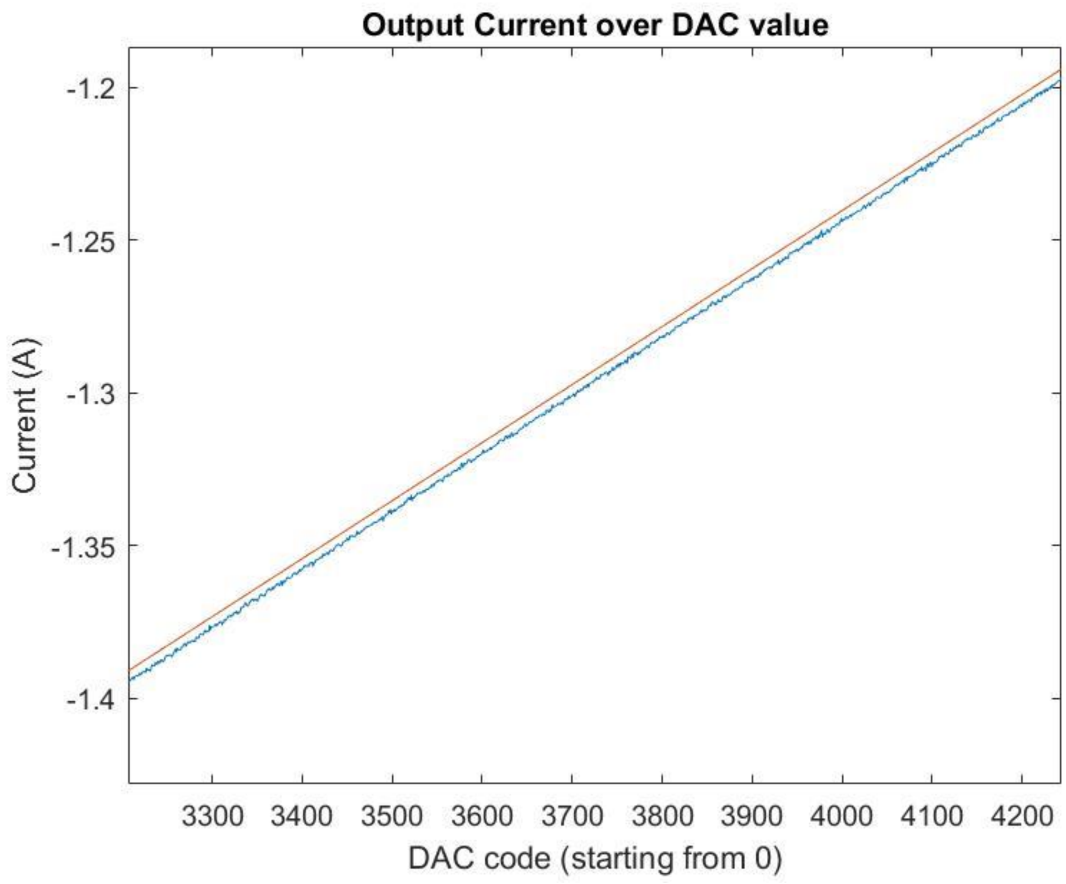




INL
×10^-3
6
5
4
3
2
1
0
-1
-2
-3
-4
Current (A)
0
0.5
1
1.5
2
2.5
DAC code (starting from 0)
×10^4



Output Current over DAC value
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Current (A)
0
0.5
1
1.5
2
2.5
DAC code (starting from 0)
×10^4



Matlab script for analysis

```
data=importdata('putty.log');
dactextvals = cell2mat(data.textdata(:,1));
dactextvals = dactextvals(:,3:6);
dacvals = hex2dec(dactextvals);

adctextvals = cell2mat(data.textdata(:,3));
adctextvals = adctextvals(:,3:6);
adcvals = hex2dec(adctextvals);

adcampvals = data.data;


[counts,centers] = hist(adcampvals,100);
figure(1)
bar(centers,counts);
DNL = counts/mean(counts) - 1;
bar(centers,DNL)
INL = cumsum(DNL);
```

```
bar(centers,INL)
title('INL from Histogram')
xlabel('Output Current (A)')
figure(2)
plot(adcampvals(:))
INL_source = adcampvals;
hold on
perfect_line=(INL_source(length(INL_source))-
INL_source(1))/(length(dacvals)).*((1:length(dacva
ls))') + INL_source(1);
plot(perfect_line);
title('Output Current over DAC value')
xlabel('DAC code (starting from 0)')
ylabel('Current (A)')
hold off

%calculate INL
figure(3)
INL=perfect_line - INL_source(:);
plot(INL)
title('INL')
xlabel('DAC code (starting from 0)')
ylabel('Current (A)')

figure(4)
INL_source = adcvals;
perfect_line=(INL_source(length(INL_source))-
INL_source(1))/(length(dacvals)).*((1:length(dacva
ls))') + INL_source(1);
INL=perfect_line - INL_source(:);
plot(INL)
title('INL normalized to LSB')
xlabel('DAC code (starting from 0)')
ylabel('ADC value')


% check for missing codes on input
for i=1:length(dacvals)-1,
    if not(dacvals(i) == (dacvals(i+1) - 1)),
        sprintf('missing code %d %x\n',i,
dacvals(i))
```

```
    end
end
```