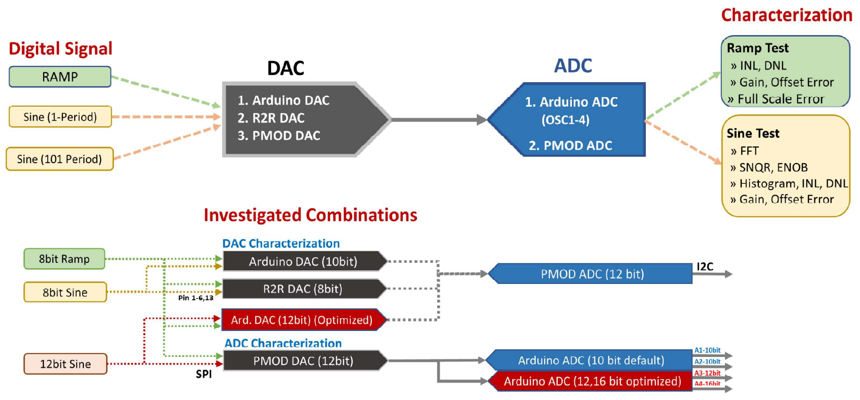
Overview



1. Goals

- Setting up of an Arduino MKR 1010 WIFI microcontroller and interfacing external data converters, analogue to digital converters (ADC) and digital to analogue converters (DAC).
- Investigation and comparison of internal and external data converters by ramp test and sine test at different frequencies (number of periods).
- Characterization of data converters for INL, DNL error, gain error, offset error, full scale error, signal to quantization noise ratio (SQNR) and effective number of bits (ENOB) by histogram and FFT (for sine signal).
- Configuring the Arduino internal data converters i-e DAC (10bit default) and 4 ADCs (10bit default).
- Interfacing the external 12-bit Pmod DAC and Pmod ADC via SPI and I2C protocols, respectively.
- Building 8-bit R2R DAC on board using resistors and investigating with 12-bit Pmod ADC.
- Investigation of 10-bit Arduino DAC and 8-bit R2R DAC with 12-bit PMOD ADC.
- Optimization of 10-bit Arduino DAC to 12-bit and comparison with 12-bit PMOD DAC.
- Investigation of 10-bit Arduino ADCs (A1 and A2) using 12-bit Pmod DAC.
- Optimization of 10-bit Arduino ADCs to 12-bit (A3) and 16-bit (A4) and comparison with 12-bit PMOD DAC.
- Spectral analysis by different applying windows on unsynchronized test signals (sketch with 256 codes having jitter).
- Documentation of project report on Webpage using HTML and JavaScript.

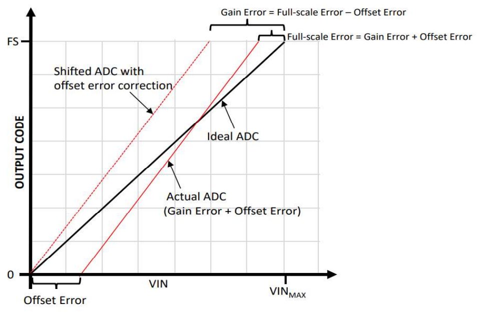
2.1 Data Converters

All the real-world quantities are analogue in nature. We can represent these quantities electrically as analogue signals. An analogue signal is a time varying signal that has any number of values for a given time slot. Opposite to that, a digital signal varies suddenly from one level to another level and will have only finite number of values for a given time slot. The electronic circuits, which can be operated with analogue signals are called as analogue circuits. Similarly, the electronic circuits, which can be operated with digital signals are called as digital circuits. A data converter is an electronic circuit that converts data of one form to another.

2.1.1 Analogue to Digital Converters

An analogue to digital conversion changes an analogue signal to a digital signal. The interfacing circuit that converts the analog signal into digital signal is called as Analog to Digital Converter abbreviated as ADC. An example for a ADC converter is microphone that converters the analogue sound input to a digital signal. Analogue signals have continuously changing values which come from various sources and sensors which can measure sound, light, temperature or movement, and many digital systems interact with their environment by measuring the analogue signals from such transducers. Basically, an analogue to digital converter snaps an analogue voltage at one instant in time and produces a digital output code which represents this analogue voltage. The number of binary digits, or bits used to represent this analogue voltage value depends on the resolution of an A/D converter. For example, a 4-bit ADC will have a resolution of one part in 15, (2^4 – 1) whereas an 8-bit ADC will have a resolution of one part in 255, (2^8 – 1). Thus, an analogue to digital converter takes an unknown continuous analogue signal and converts it into an “n”- bit binary number of 2^n bits [3].

There are 5 ADCs used in this project.
4 arduino integrated accessed at Pin1-4 each with deafult 10 bit resolution and one 12 bit Pmod external ADC.
The Pmod ADC is used for the characteriation of the 8 and 10 bit DACs that are introduced in next subsection.
Two of the arduino ADCs were further optimized to 12-bits and 16-bits.


Gain Error = Full-scale Error – Offset Error
Full-scale Error = Gain Error + Offset Error
FS
Shifted ADC with offset error correction
Ideal ADC
OUTPUT CODE
Actual ADC (Gain Error + Offset Error)
0
Offset Error
VIN
VIN_MAX

2.1.2 Digital to Analogue Converters

Similarly, if we want to connect the output of a digital circuit as an input of an analogue circuit, then we have to place an interfacing circuit between them. This interfacing circuit that converts the digital signal into an analogue signal is called as Digital to Analogue Converter. An example can be a speaker than converts a digital coded sound signals to an analogue sound audible by our ears.

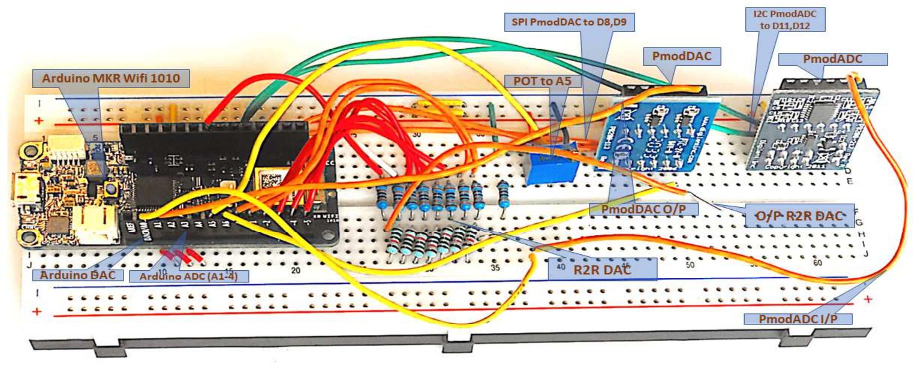
There are three DACs used in this project.
1. A 8-bit R2R ladder DAC
2. 10-bit Arduino's DAC (at pin A0). It was furhter optimized to 12-bits.
3. 12-bit External Pmod DAC (used for ADC characterization and comparison with optimized 12-bit arduino DAC.

2.1.3 Characterization Parameters

ADCs introduce static errors to measurements [4-5]. Static parameters include offset error, full-scale error, gain error, and total unadjusted error. Nonlinear errors include integral non-linearity error and differential non-linearity error.
Offset error (minimum code) is the difference between the actual first transition voltage and the ideal first transition voltage. Ideally, first transition takes place at a voltage equal to ½ LSB (least-significant bit).
Full-scale error (maximum code) is the difference between the actual last transition voltage and the ideal last transition voltage. Ideally, last transition takes place at ½ LSB before the rail voltage.
The resolution of an ADC is usually expressed as the number of bits in its digital output code. For example, an ADC with an n-bit resolution has 2^n possible digital codes which define 2^n step levels. However, since the first (zero) step and the last step are only one half of a full width, the full-scale range (FSR) is divided into 2^n – 1 step widths. Hence,
1 LSB = FSR / 2^n – 1 for an n-bit converter
For a DAC, 1 LSB corresponds to the height of a step between successive analog outputs, with the value defined in the same way as for the ADC.

3. Experimental Setup (Arduino Environment)

Breadboard Diagram | Original Arduino Setup | Circuit Schematic Diagram



3.1 Hardware Setup

An Arduino MKR 1010 Wifi microcontroller has been used to exploit its internal ADCs and DACs and also interfacing external high resolution Pmod ADC and DAC. In the results section Arduino MKR is often referred as just MKR as well. Arduino MKR 1010 is 32-bit microcontroller and has 3.3V operating voltage and 5V input voltage. The controller was powered by the USB port of the laptop. The MKR 1010 has 13 PWM pins, out of which 8 pins has been used for 8-bit R2R ladder DAC. There is one SPI and one I2C serial communication protocols, the SPI has been used for interfacing Pmod DAC while the I2C has been used for interfacing Pmod ADC. The MKR has one 10-bit default DAC for analogue output that is accessible at pin A0. Analogue pins A1-A4 are used as ADCs and A5 is used to take input from potentiometer (POT) to adjust the frequency (number of periods). The MKR's VCC and GND are used for input voltage and ground for external Pmod ADC and DAC, R2R and POT. A detailed schematic diagram is shown in the Schematic Diagram section of report. The PMOD ADC is configured to take input signals at V1 channel only, and therefore only one input is (R2R, or MKR DAC) connected at a time. This connection is represented with dotted lines the schematic and breadboard diagrams shown above. The MKR's ADC 1-4 (pin A1-A4) are shorted to receive the same analogue signals. The configuration is further discussed in the next section.

3.2 Arduino Sketch

8 digital pins (0-6,13) are defined for 8-bit R2R input. The R2R resolution can increased by using microcontroller that has more digital pins that can connect more resistors. A DAC with higher resolution can be developed by utilizing the PWM (like pin 7) but it requires external resistor and capacitor circuit to realize a DAC. Similarly, the pins A1-A4 are defined as analogue input (ADCs). The analogue input from A6 is stored in integer variable freq [uint16_t freq = analogRead(ADC_FREQ)] and the number of periods is decided by the condition if/else. However, in the final version of sketch the POT has been bypassed and integer value of the freq is manually entered to select the number of periods. In this project 1 period sine (Sine1) and 101 period sine is selected. Sine lookup table are implemented for generating sine wave for sine test. The Pmod ADC is configured fro I2C [write.begin();] and the Pmod DAC is configured for SPI [SPI.begin();] at clock frequency of 1MHz. A 12 bit sine signal is generated by lookup table and truncated to 8 bit (sine8 = (sine12>>4)&0XFF;) for 8-bit R2R DAC and arduino DAC [analogWrite(DAC, sine8)]. The 12-bit sine input is given to Pmod DAC for ADC characterization, [writeDAC(sine12);, where writeDAC(); is a function defined for Pmod DAC]. The sketch was modified for Arduino MKR DAC to analogWrite(DAC, sine12)] for checking the optimized 12-bit DAC. The data from all 5 ADCs is printed in separate columns on the serial monitor for data logging and further analysis [Serial.print(oscX);].

4.1 Investigated combinations

As shown in the overview section, different converter combinations are investigated in this experiment. The measuring converter is kept higher compared to the converter under test i-e while investigation of DACs the external 12-bit Pmod ADC is used and for investigation of Arduino ADCs the external 12-bit Pmod DAC is used. A DAC with higher resolution can generate a suitable high-resolution signal to test ADCs and a high-resolution ADC can sample effectively the output of DAC.

Therefore, for DAC characterization: R2R DAC and Arduino MKR's DAC is tested with Pmod ADC. While characterization of ADC: The Arduino ADCs are characterized by input test signals from external Pmod DAC.

Summarizing the combinations;

1. R2R DAC with PmodADC
2. Arduino MKR 10-bit DAC with PmodADC
3. Arduino 10-bit ADC (A1 & A2) with Pmod DAC

The ramp test for the DAC is performed by the provided ReadOscilloscope Tool. For the sine test, it was aimed to generate an ideal sine wave and compare it with the real since data and get INL, DNL. But this couldn't be implemented. However, the FFT tool was used to characterize the DAC and for later comparison with the ideal sine wave implementation.

Although the comparison of DAC sine test FFT couldn't be compared with ideal sine but the results are still kept in the report for future refence and data for interpretation of existing FFT tool. Furthermore, the SQNR was not changing in the FFT tool when it charts were generated for lower number of bits during attempts to reduce INL, DNL. Therefore, the first and only possible values were calculated by following formulas and enlisted in the above tables.

SQNR = signal magnitude [dB] – total noise magnitude [dB]

ENOB = (SQNR-1.76)/6.02

Since there were no final values of ENOB, the rated no of bits is mentioned as the converters resolution i-e R2R as 8 bit, Arduino default 10-bit and Arduino optimized as 12-bit. These rated values were further used for used for calculation of LSB by following formula.

LSB=3.3/2n-1, where n is selected as 8, 10 and 12.

4.2 Optimization of Arduino DAC & ADC

Since Arduino DAC and ADCs are set to 10 bits by default, but their resolution can be increased by accessing them via IDE sketch. The resolution of Arduino DAC is increased from 10bits to 12 bits by using the code;

analogWriteResolution(bits);

This sketch line is written before the [analogWrite(DAC, sine12);], here sine 12 given as input to let the DAC generate signal that can be tested by 12-bit ADC and then compare with the external 12-bit Pmod DAC. Below is the modified Arduino sketch for optimization of the Arduino DAC.

```
<!-- Arduino IDE DAC optimization Sketch -->

analogWriteResolution(12);     // Arduino DAC Resolution Change
     analogWrite(DAC, sine12); //  //Writing Analogue to arduino DAC//
 >
```

Similarly, the resolution of the ADC is increased from 10 bits to 12 and 16 bits. The results of 12-bit optimized combination are realistic and included in the report. However, it was observed that Arduino MKR WIFI 1010 has maximum accessible resolution of 12 bits, although 65,536 steps (quantization) were achieved but arduino was padding the extra bits with zeros. The following addition was made in the sketch before the serial print argument to increase the ADCs resolution.

analogReadResolution(bits);

The first two ADCs were configured to default 10bit, the third ADCs was configured to 12bit and the 4th ADC was configured for 16 bit. The modified sketch is shown below.

```
<!-- Arduino IDE ADC optimization Sketch -->

     analogReadResolution(10);
           oscX = analogRead(ADC_OSC1);       //OSC1,2 10bit defalut Resolution but requires to be defined//
           Serial.print(" , OSC1-10bit = ");
           Serial.print(oscX);

           analogReadResolution(10);         //OSC1,2 10bit defalut Resolution but requires to be defined//
           oscX = analogRead(ADC_OSC2);
           Serial.print(" , OSC2-10bit = ");
           Serial.print(oscX);

           analogReadResolution(12);         //OSC3 Resolution Change to 12 bit//
           oscX = analogRead(ADC_OSC3);
           Serial.print(" , OSC3-12bit = ");
           Serial.print(oscX);

           analogReadResolution (16);        //OSC4 Resolution Change to 16 bit//
           oscX = analogRead(ADC_OSC4);
           Serial.print(" , OSC4-16bit = ");
           Serial.print(oscX);
```

5.1 DAC Characterization

5.1.1 DAC Characterization (Results)

RAMP Test - R2R DAC

Sine1 Test - R2R DAC

Sine101 Test - R2R DAC

RAMP Test - MKR DAC 10bit

Sine1 Test - MKR DAC 10bit

Sine101 Test - MKR DAC 10bit

RAMP Test - MKR DAC 12bit

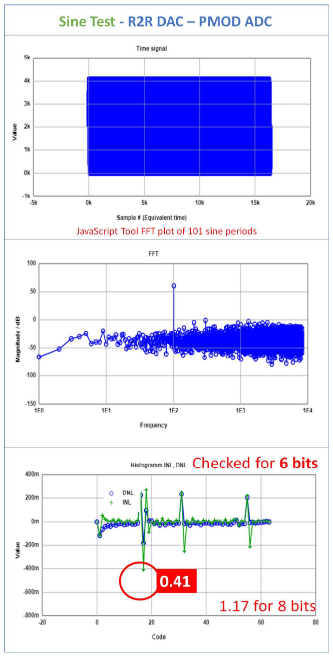
Sine1 Test - MKR DAC 12bit

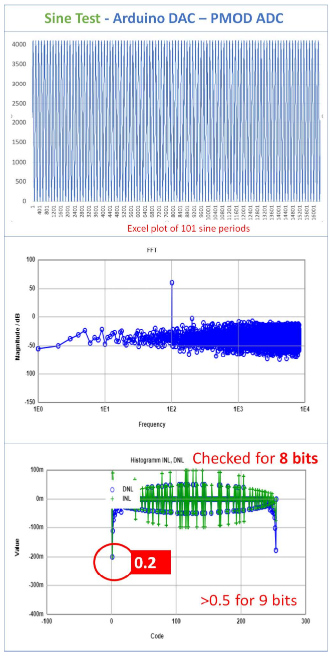
Sine101 Test - MKR DAC 12bit

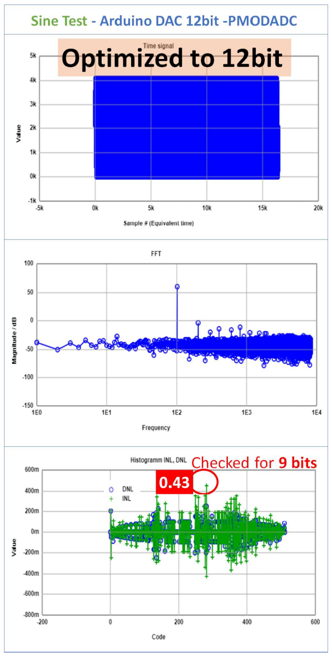
DAC Ramp Test			
Parameter	R2R DAC	Arduino MKR DAC Default 10bit	Arduino MKR DAC Optimized 12bit
INL,DNL	0.85 LSB	0.28 LSB	0.26 LSB
Offset Error	0	1	1
Gain Error	14	18	6
DAC LSB	12.94 mV	3.22 mV	0.806 mV
DAC Resolution	8 bits	10 bits	12 bits

DAC Sine Test (1 Period)			
Parameter	R2R DAC	Arduino MKR DAC Default 10bit	Arduino MKR DAC Optimized 12bit
INL,DNL - Resolution	0.41 LSB for 6bits	0.2 LSB for 8bits	0.44 LSB for 9bits
SQNR	48.72	49.51 dB	58.48 dB
ENOB	7.8 bits	7.93 bits	9.42 bits
Maximum value	4079	4079	0
Minimum value	0	0	4091

DAC Sine Test (101 Period)			
Parameter	R2R DAC	Arduino MKR DAC Default 10bit	Arduino MKR DAC Optimized 12bit
INL,DNL - Resolution	0.41 LSB for 6 bits	0.2 LSB for 8bits	0.43 LSB for 9bits
SQNR	48.70	49.52 dB	58.52 dB
ENOB	7.8 bits	7.93 bits	9.43 bits
Maximum value	4082	4082	0
Minimum value	0	0	4091







5.1.2 DAC Characterization Discussion

R2R DAC:

A greater than half LSB is observed for the R2R ramp test. The gain error was 14 but the offset error of the R2R was ideal i-e 0. The reasons for the higher DNL in R2R is because of the non-ideal components and structure as all the resistors do not have the ideal rated values and the overall R2R performance is also influences by the breadboard patched circuit. For the sine test two different periods were observed i-e 1 and 101. In the16k code there was no jitter in the input sine signal and therefore negligible differences are observed both for the sine with 1 and 101 periods. The ENOB for R2R was calculated as 7.8 but for this value the INL, DNL were >0.5. Further reducing the bits in the FFT tool, reduced the INL, DNL and the acceptable (0.41) was achieved for 6 bits.

Arduino MKR 10-bit R2R:

For Arduino MKR, the INL, DNL is less than 0.5 LSB i-e 0.28. This shows that MKR 10-bit DAC is better than R2R DAC as the real ramp data is in the acceptable range from the ideal data. The same was observed for both ramp and sine test with sine 1 and 101 period signal. For the 10-bit DAC the acceptable INL, DNL was achieved at 8 bits while is 6 for R2R.

12-bit Optimized Arduino DAC:

In order to achieve higher resolution for 10-bit Arduino DAC, it was optimized as explained in the experimental procedure section. The INL, DNL of ramp test were further reduced to 0.26 for the optimized ADC. Although the sine test for DAC via FFT tool was not recommended, but results are presented for an idea how it turns out and might be seen and compared when the tests are performed with ideal sine comparison with real sine signal. Here it was observed the acceptable INL, DNL was achieved at 9 bits. However, for default 10-bit Arduino DAC, the INL, DNL were greater than 0.5 when FFT was performed even the missing codes are zero at 10-bits. So here we can observe the impact of optimizing the Arduino DAC from 10 to 12 bits. Furthermore, when comparing Arduino optimized 12 bit DAC with external Pmod 12 bit DAC it was observed that Pmod DAC shows much better performance with lower INL, DNL. Results of Pmod DAC with Pmod ADC are mentioned in the ADC characterization results.

5.2.1 ADC Characterization (Results)

ADC Ramp Test				
Parameter	MKR ADC-1 10bit	MKR ADC-2 10bit	MKR ADC-3 12bit optimized	PMOD ADC 12bit
ADC Resolution	10 bit default	10 bit default	12 bit optimized	12 bit
INL,DNL	0.55 LSB	0.52 LSB	0.3 LSB	0.26 LSB
Offset Error	1	2	9	3
Gain Error	3	3	12	3
ADC LSB	3.22 mV	3.22 mV	0.806 mV	0.806 mV

ADC Sine Test (1 Period)				
Parameter	MKR ADC-1 10bit	MKR ADC-2 10bit	MKR ADC-3 12bit optimized	PMOD ADC 12bit
INL,DNL - Resolution	0.2 LSB for 7bits	0.2 LSB for7 bits	0.34 LSB for 8bits	0.43 LSB for 9bits
SQNR	54.2	54.22	55.17	63.26
ENNOB	8.71	8.71	8.87	10.22
Maximum	1023	1023	4086	4094
Minimum	0	0	0	1

ADC Sine Test (1 Period)				
Parameter	MKR ADC-1 10bit	MKR ADC-2 10bit	MKR ADC-3 12bit optimized	PMOD ADC 12bit
INL,DNL - Resolution	0.22 LSB for 7bits	0.17 LSB for 7 bits	0.34 LSB for 8bits	0.39 LSB for 9bits
SQNR	53.9	53.96	54.72	63.29
ENNOB	8.66	8.67	8.8	10.22
Maximum	1023	1022	4087	4094
Minimum	0	0	0	1

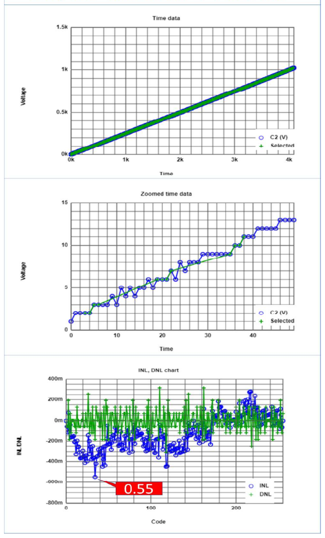
Please select test for Arduino ADC 1 (10-bit):
[✓ Ramp Test]
Sine Test (1-Period)
Sine Test (101-Period)

Please select test for Arduino ADC 2 (10-bit):
Ramp Test
[✓ Sine Test (1-Period)]
Sine Test (101-Period)

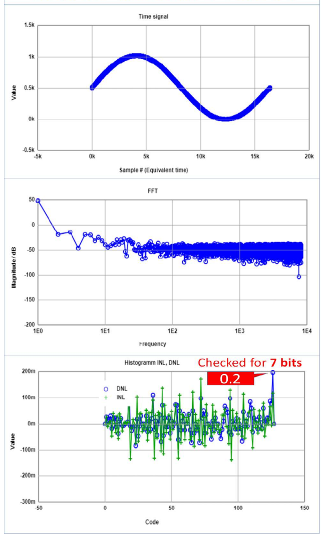
Please select test for Arduino ADC 3 (12-bit):
Ramp Test
Sine Test (1-Period)
[✓ Sine Test (101-Period)]

Please select test for PMOD ADC (
Ramp Test
Sine Test (1-Period)
[✓ Sine Test (101-Period)]

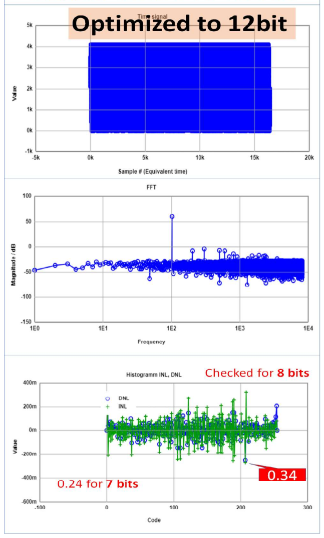
Ramp Test - Arduino ADC-1 – PMOD DAC



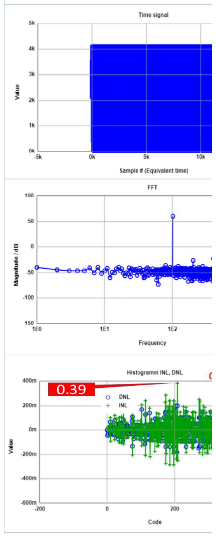
Sine (1) Test - Arduino ADC-2 – PMOD DAC



Sine (1) Test - Arduino ADC-3 – PMOD DAC



Sine (1) Test - PMOD ADC – P



5.2.2 ADC Characterization Discussion

Four different ADCs were tested with external PmodDAC having 8bit ramp and 12-bit sine input signal. The issue for the FFT output table was encountered i-e by decreasing the number of bits and regenerating the FFT charts, the INL, DNL was reducing but the signal and noise magnitudes were not changing, Therefore, here also the ENOB was is calculated by the first and only available data and reported. While the bits at which the INL, DNL <0.5 is achieved, are mentioned in the ADC results tables as INL, DNL-Resolution.

As explained in the experimental procedure section that the first two (A0, A1) ADCs of Arduino are configured to default 10 bits resolution. The 3rd ADC (A3) is optimized to 12 bits and here in the results it compared to 12-bit Pmod ADC.

For the ramp test, the INL, DNL was around 0.5 for both 10-bit ADCs, however, after for optimized ADC, the INL, DNL is 0.3 LSB. Which is close to the 0.26 from external Pmod ADC. However, like Pmod DAC, the 12-bit Pmod ADC has also performed better and have lower INL, DNL compared to the 12-bit optimized Arduino ADC.

Sine test for the ADC is also performed for two different number of periods i-e 1 and 101. Since the 16k code that is generated for 12-bit data converters doesn't has jitter, so the same results were observed for 10-bit ADCs for both 1 and 101 period sine signals. For FFT at 7 bits the INL, DNL was achieved in the acceptable range i-e 0.2.

Comparing the 12-bit optimized Arduino ADC with 10-bit default ADCs, it was observed that <0.5 INL, DNL was achieved at 8 bits instead of 7 bits. For 10-bit ADCs the INL, DNL was >0.5 for 8 bits. Therefore, the optimization has increased the resolution of Arduino ADC. Further, comparing the optimize Arduino ADC with the Pmod ADC, it was observed that Pmod ADC has even better performance and <0.5 INL, DNL is achieved at 9 bits. The SQNR and ENOB of the optimized Arduino ADC is higher than the default 10-bit ADCs, however the SQNR and ENOB of PmodADC is significantly higher compared to the Arduino optimized ADC.

Please select the window:

[✓ No Window (Rectangular)] Hamming Window Kaisser Window Nutall Window

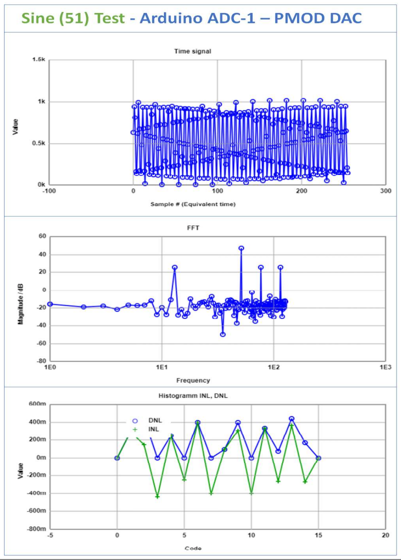
<

5.3 Windowing

The deviation of the time instant between two ADC samples is known as the clock jitter. To achieve the optimal system and data converter performance the frequency-domain mechanisms of jitter-induced sampling errors should be considered. While increasing the sampling frequency the clock jitter increases conversion noise, which reduces the overall system performance. The 16k code is observed to be jitter free and has synchronized clock signal and therefore no jitter effect was observed.

However, for 256 value code, the jitter was introduced and increase in noise is observed in the FFT spectrums. FFT spectrum of a given discrete-time signal is performed to analyze the presence of sinusoids and their frequency. Ideal FFT is obtained when integer number of periods of a periodic signal are measured. Since, practically full length DFT is not feasible because of speed and storage limitations, therefore the duration of the signal is limited to the time interval T0 = NT, where NL is the number of samples and T is the sample interval. Limiting the signal $x[n]$ to N samples in interval (0 ≤ n ≤ N-1) is equal as multiplying $x[n]$ by a rectangular window $wr[n]$ of length N. So now here we can define window as a function that is non-zero in a finite range of time4 and windowing truncates a signal to that finite length and reshapes it2. Precisely, the window function modifies the amplitude and frequency response of the sampled data5. The rectangular windowing is mare truncation without any reshaping.

Two main parameters to evaluate our window are; (i) width of the main lobe with reference to the width of rectangular window of same length (its $4\pi/N$ for rectangular window and good), (ii) level of largest side lobe with reference to the amplitude of main lobe (it is -13.5.dB for rectangular window and undesirable). By windowing, our desire is to decrease the width of main lobe and increase the flatness of side-lobe level and either one is obtained over the cost of other, so we need to compromise and it depends on the type of application, e.g5 Blackman is considered best for ADC characterization. Here 4 different windows have been applied and the it is observed that Kaiser window has served best as after applying the Kaiser window the frequency of the main signal has become more visible and can be distinguished because of noise added by jitter.



6. Challenges

- Sine Test for DAC remained a challenge because of curve fitting of ideal sine wave requirement with adjusted offset amplitude, and phase.
- Simultaneous connection of Arduino ADCs and PmodADC influences the readings of Arduino ADCs and higher INL, DNL values were observed. However, the reading of the Pmod are not influenced regardless of the Arduino ADCs. The new readings were taken after removing the PmodADC.
- Long wires between converters and jumpers also reduce the quality of the signal conversion. The R2R INL, DN was greater than 1.2 but after connecting the R2R directly with one single wire with PmodADC the INL, DNL was drastically reduced to 0.85.
- The precision of data logging from ADCs is not so high as under same test conditions different readings are obtained on repeating the experiments. Each reading was taken thrice to increase the experiment precision.
- The ADCs optimization requires the resolution definition for each ADC separately even when configured for default 10-bit. Configuring only one ADC for 12 bit without writing code for each ADC, all the ADCs start to give 12 bit values. Therefore, when alternation is required, the resolution of each ADC should be defined separately even when default resolution of 10 bit is required. The sketch images are shown in the adjacent image on the right side.
- Documenting all the graphs and values in a concise way was the biggest challenge for me in this project. The Prof. Vollrath's provided toggle button for swapping tables in the web report sample and changing multiple images in same place from microelectronic lectures inspired me to write JavaScript function for buttons to present more data concisely. As the number of buttons increased, radio buttons have been implemented by writing another script with switch-break so that the reader can keep a track on which graphs he is currently looking at.

7. Summary

Any data converter can be characterized by ramp and sine tests and analyzing the static (offset and gain errors) and nonlinearity (DNL, INL) errors. The converter that is used to test the CUT (other device/converter under test) must have higher resolution then that CUT. In this characterization project, a 12-bit external Pmod ADC is used when CUT was 8-bit R2R, 10-bit Arduino integrated DAC. Likewise, 12-bit external Pmod ADC is used when the CUT is 10-bit Arduino ADC. It was observed that the performance of R2R DAC is limited because of its low resolution, non-ideal values of resistors and breadboard patching. The Arduino converters were further optimized for higher 12-bit resolution via modifications in the sketch. It was observed that optimized converters perform better than default 10-bit converters but when compared with the equivalent 12-bit Pmod converters, the performance of the Pmod converters is higher. Both low and high frequency sine tests are performed via two different codes with and without jitter. It was observed that higher frequency signals cause clock jitter in the ADC and increases the noise in the FFT reducing the effective number of bits. Different windows were applied to improve the FFT spectrum and make the main signal component visible from noise. From learning perspective, I believe that I can build a test setup for data converters and I can perform their ramp and sine tests. I can characterize the converters by ramp static and nonlinearity errors and sine histogram and FFT. The tests can be performed both offline on excel by generating ideal ramp and sine signals and finding INL, DNL by comparison with the real values, and online by utilizing Prof. Vollrath's ReadOsc and FFT tools which are far more accurate, concise and time efficient.

8. Quick Links and Data Download

Please click to downlaod the View IDE sketch used in this project.
View Arduino IDE Sketch

Please click to downlaod the original IDE sketch for further programming.
Download Sketch

Please click to downlaod the logged data for all the converter combinations tested.
Converters Data Logging

9. References

[1] Making of a Webreport , Vollrath

[2] Interface Electronics Laboratory Manual for ADC and DAC Characterization in Arduino environment , Vollrath

[3] Interface Electronics Lecture Notes , Vollrath

[4] Brückl, S., 2020. DSP Lecture Notes and Lab Manuals, M-EE UAS Kempten

[5] Understanding data Converters, application report , Texas Instruments

[6] Compensating for DAC Offset and Gain Error , Analogictips

[7] ADC Gain and Offset Error Calibration on ARM® Cortex®- M0+ Based MCUs , Microchip