Investigation on Atmega328 Accuracy

Table of Contents

1. Overview

A while back I ran some tests with sinewaves, to get an idea of the accuracy of the ADC on an Arduino UNO. The results did not look right, so I tried to get an in-depth understanding of what is going on. I came up with two test setups, one for testing with sinewaves, and one for exercising a DAC/ADC combination. The first circuit is actually a preamplifier for the ATmega ADC, with an input which is compatible with standard oscilloscope probes. I build these circuits on solderless breadboards, using the ATmega328-PU in DIL package, which is only slightly different from the Atmega328P-PU of the Arduino UNO. I do not need the Arduino board, but use the coding infrastructure of GNODUINO.
For my investigations I transfer most of the data from the ATmega device to the PC, through the serial interface. There I use two programs for displaying and analyzing the data:

- LXARDSCOPE: two-channel oscilloscope, with data rate close to 3000 samples per second.
- LXSNDTEST: originally created for testing the properties of soundcards, but then extended to include testing of ATmega type devices. Uses sinewaves and FFT for processing.

I refer to both programs in the following chapters as appropriate; both programs are available from Sourceforge (LINUX only).
I also gave some thought to the communication interface between the ATmega device and the PC. While serial ports are being phased out in favor of USB, they have one distinct advantage: the relatively low data rates allow for inserting optocouplers into the signal path, which enables complete electrical insulation of the PC from the ATmega device with its associated circuitry. Since all the setups described below include such an interface, I describe this circuit first.

2. Communication with PC

Hardware issues:

- the Atmega328 series come with a serial port only
- there is no ATmega device with USB interface in a hobbyist-friendly DIL package
- today's laptop computers do not have serial ports any more
- some applications require complete electrical insulation from the PC.

To satisfy the different needs, I worked with three circuits:

- connect to serial port on PC
- connect to USB port
- insulating interface with optocouplers

Figure 1 shows the details for the three circuits. In the top right section, a serial interface with MAX232 is shown. I run the circuit with signal levels of +/-4V instead of +/-8V, to save power and to reduce noise. The 5V supply for this circuit can be shared with the ATmega device, except if the insulating interface is used.
In the course of this project I found a cheap Serial-to-USB board (www.betemcu.cn), using the CP2102 chip. To allow for programming the Atmega328 through the pins at the edge of the board, I cut the trace from the chip to the RESET pin and added a wire from the spare hole for DTR to the RESET pin.

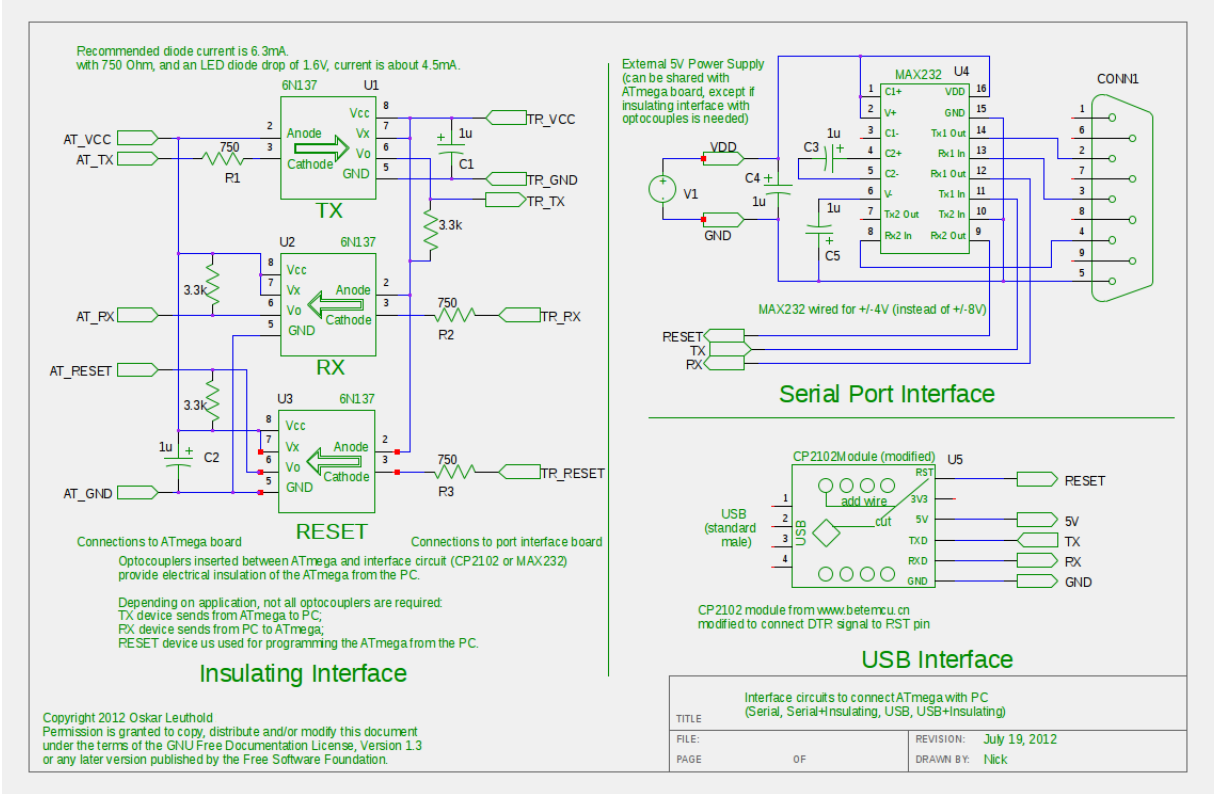

Figure 1: Interface circuits for ATmega328

The left side shows the insulating interface with three optocouplers, which allow for full communication with the ATmega device: transit, receive and program. I have used this interface successfully at 115200 baud. For many applications, only a transmit or/and a receive path might be required, so only part of the interface needs to be built.
Please review the datasheet for the optocoupler carefully if you need high voltage separation between the two sections, and be careful with your pc-board layout and/or wiring.

In all my programs, data transfer from the ATmega to the PC is designed with two words, each with 10bits. The data is sent in groups of four bytes, arranged as follows:

Byte 1: bits 5 to 9 (MSB) of word #1 + 224 (codes 224 to 255)
Byte 2: bits 0 (LSB) to 4 of word #1 + 96 (codes 96 to 127)
Byte 3: bits 5 to 9 (MSB) of word #2 + 96
Byte 4: bits 0 (LSB) to 4 of word #2 + 96

Byte 1 is the marker for the group. This proprietary arrangement is compatible with LXARDSCOPE and LXSNDTEST.

3. Sinewave Test with External Source

The program LXSNDTEST on the PC is used in the following setup:

- create a high purity sinewave with a soundcard,
- feed sinewave into the ATmega's ADC,
- collect the digitized samples from ATmega,
- process samples with FFT and display results.

This setup produced unsatisfactory results. While investigating this problem, I created a universal preamplifier with the following characteristics:

- input impedance 1MegOhm || 33pF, the oscilloscope standard. This allows for standard oscilloscope probes to be used with attenuation of 10x and 100x for higher input voltages.
- input voltage range from 0 to 5V, same as the full scale range of the ADC.
- input not connected directly to an IC, to avoid stressing the input protections when inadvertently too high an input voltage is applied, as it can happen easily when investigating electronic appliances.

I also thought about designing a differential input, but the measurement results seem to indicate that power supply noise is not an issue with the kind of accuracy I am looking for. Also I could not find a simple circuit which would allow for an input voltage range of +/-5V with a single 5V supply, without exhibiting some unusual behavior, like pulling up to VDD instead of down to GND.
I found some MC33204 circuits in my stock, so I tried them first. The following schematic shows the setup with these bipolar quad opamps.

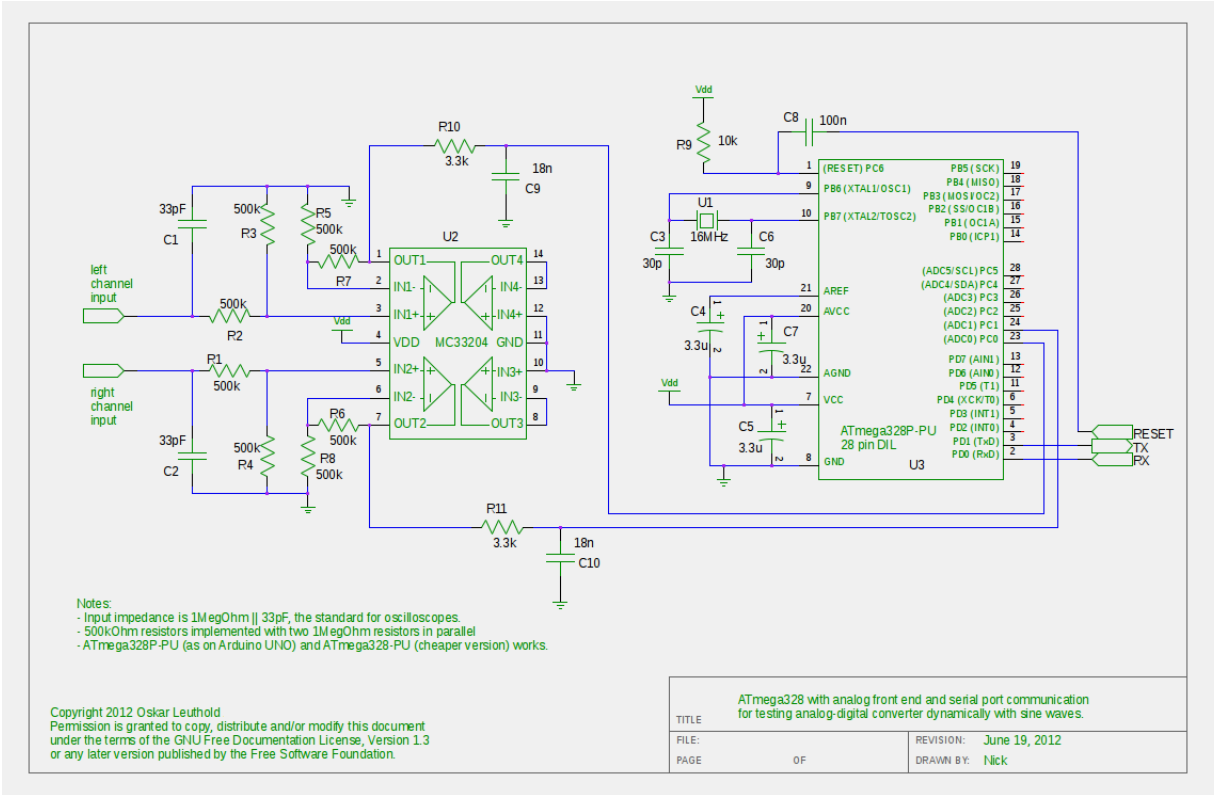

Figure 2: ATmega328 with MC33204 in input buffer

The input signal is divided by two, before it gets into the opamp, which is configured with a gain of two, thus the net gain from input to output is one. The input amplifier needs to have an input voltage range of GND to 2.5, and output voltage range from GND to VDD (rail-to-rail). Resistance from

inverting and non-inverting inputs to GND is equal, thus input current is not a big issue, just the offset. While investigating differences in performance of the left and right channel, I observed higher distortion in amplifier #3, compared to the others.
I built the circuit on a solderless breadboard, with an ATmega328-PU, which is a cheaper version of the circuit used on the Arduino UNO board. For communication with the PC, I use the serial port with a MAX232, as shown in figure 1.
I built a second version once I found Microchip's MCP6002, which has two opamps in an 8 pin DIL package. This chip works just fine, with identical results for both channels.

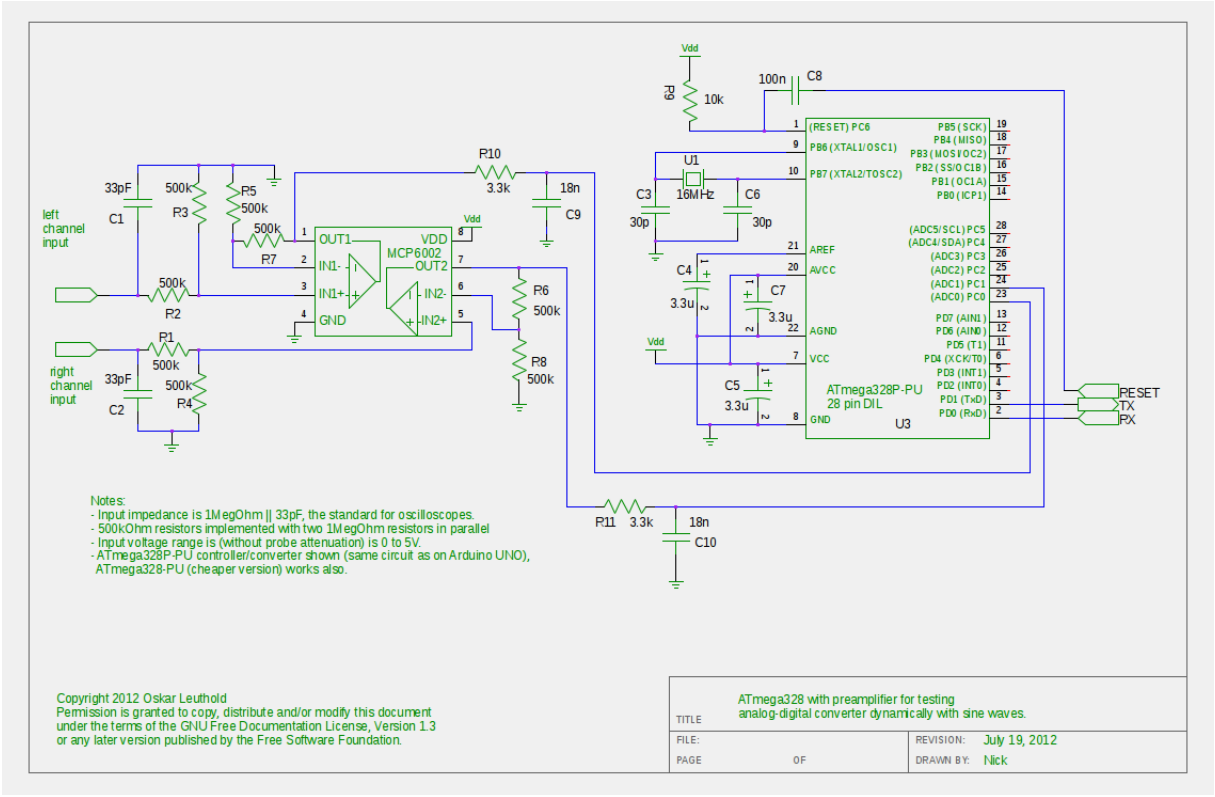

Figure 3: ATmeg328 with MCP6002 input buffer

The program lxardoscope.pde for the ATmega samples the two channels continuously and sends the samples through the serial port to the PC, as described in chapter 2. The program uses Arduino constructs throughout and was published a while back in connection with LXARDSCOPE.
With a sinewave from an Extigy soundcard, which is good to about 13 bits, I got the following results (which are the same as for the MC33204, once the problem with the third amplifier was identified):

	left	right	
Minimum input signal (lsb)	38	34	
Maximum input signal (lsb)	996	994	
Fundamental level	50.6	50.6	dB
THD	0.050	0.047	%
ENOB	9.0	9.0	
SNR	56.3	56.0	dB
SINAD	55.8	55.6	dB
SFDR	61.9	61.7	dB
@freq	1371	1370	Hz

The sampling rate is close to 2941. The reported spur at 1370 is just at half the sampling rate.

The following screenshot shows the setup and spectrum result from LXSNDTEST:

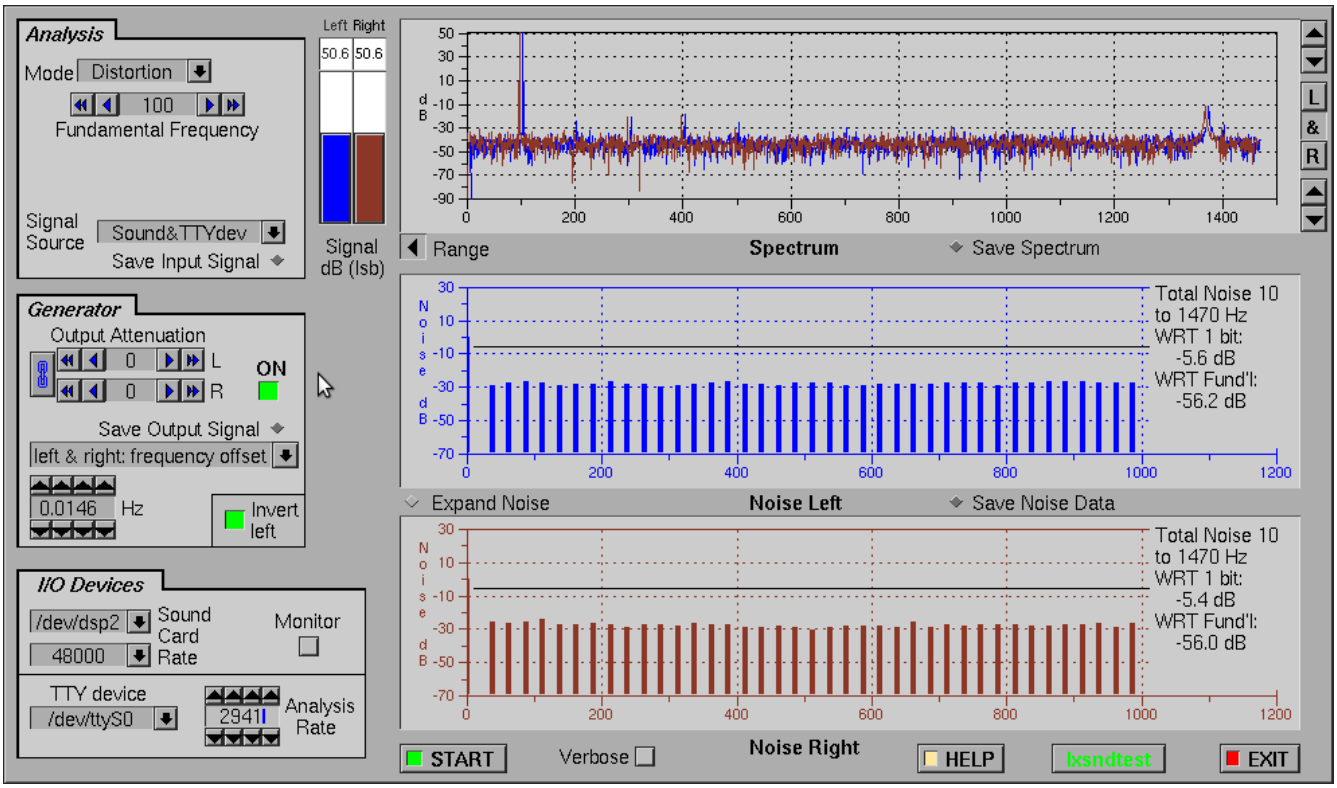

Figure 4: Spectrum ATmega328 with MCP6002, 100Hz sinewave input from soundcard

The harmonics of the 100Hz signal and the spur at 1370 can easily be seen on the spectrum plot. Note that in between, the noise level is around -45dB, or -95dB with respect to full scale.
(Your setup will be different, in particular for “Analysis Rate” and “left & right: frequency offset”. See the LXSNDTEST documentation for details.)

An important element in the signal chain is the RC lowpass filter connected between the buffer output and the ATmega input. It looks like the switched capacitor circuit, which forms the input stage of the ADC, works best when it connects to a relatively large capacitor, probably because of charge injection issues. The following table reflects this observation:
(ENOB0 is with 0 phase, ENOB180 with 180 phase between left and right channel)

R	C	ENOB0	ENOB180	-3dB point
0	0	8.4	8.4	
3.3k	18n	9.0*	8.9	2.7kHz
3.3k	5n	8.9	8.9	
3.3k	1n	8.7	8.6	
3.3k	0	8.4	8.4	
10k	18n	8.9	8.9	880Hz
10k	5n	8.9	8.9	3.2kHz
10k	1n	8.9	8.4**	
10k	0	8.4	8.4	

* detailed results reported above.

** no explanation available for difference in channel characteristics

Another reason is simply that a properly designed the lowpass filter reduces high frequency noise

above the sampling frequency of the ADC. In our case, the sine wave from the soundcard contains noise components above 2940Hz, which get all aliased down into the frequency range from 0 to 2940Hz.

4. Setup for DAC/ADC combination

For the next series of tests, the ATmega device is paired with a DAC. MAX503 is a 10 bit converter with 5V supply and internal reference, in a DIL package, for a reasonable price. The following schematic shows how the two devices are connected:

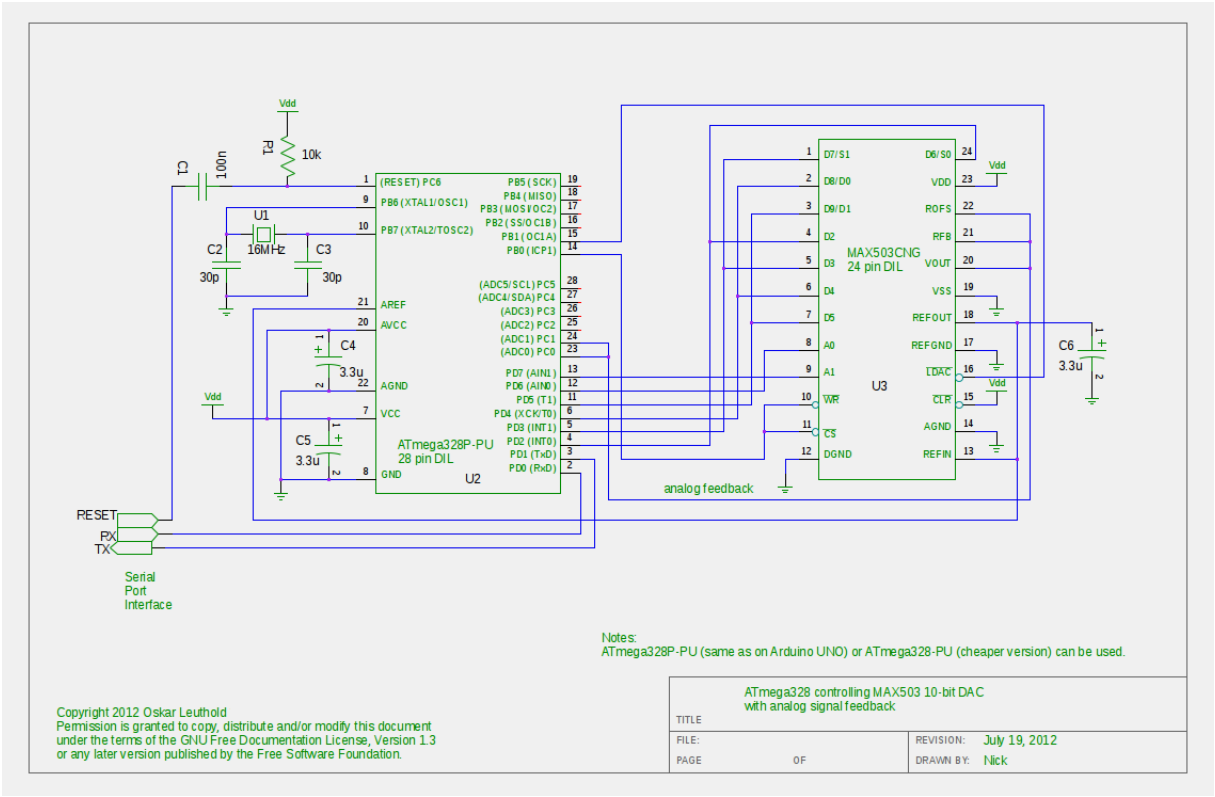

Figure 5: ATmega328 with MAX503 in feedback configuration

The ATmega outputs the code for the DAC levels on the four lines PD2, PD3, PD4 and PD5, and control signals on PD6, PD7, PB0 and PB1. The ATmega receives the reference voltage on AREF, and the analog signal from the DAC output on pins PC0 and PC1.
In the ATmega programs written for this setup, a subroutine (`void dac(int v) { …}`) exercises the control lines while transferring the 10bit control code in 3 nibbles from PORTD to the MAX503.

5. Ramp Test with DAC/ADC Combination

The idea of this test is to produce a triangular wave for exercising all the codes, incrementing from 0 to 1023, then decrementing back to 0. Ramping in both directions allows for finding hysteresis effects. With the measured data, two standard measures for ADCs and DACs can be calculated: Integral and Differential Nonlinearity.
The sketch ramp.pde is used to control the ATmega device. The sketch makes use of Arduino macros, except in the subroutine which controls the DAC. At the beginning, ramp.pde describes also the wiring between the ATmega and the MAX503. An LED+resistor connected to pin 17 serves as pulse indicator.

The loop() section of the program generates the ramp up (with pin 17 high) and ramp down (with pin 17 low). The ATmega device outputs on its serial port the 10-bit digital reference value of the ramp, and the value measured by its ADC, for processing in the PC. This setup of course tests the combined DAC and ADC characteristics of the two devices. If any problems were found, then additional tests would have to be performed to identify which of the two devices is the major contributor.

The companion ramp.c program on the PC collects data from the serial or USB port, waits for the beginning of the ramp up phase, and then processes a preset number of full cycles. The program checks for the distribution of the measured values, separately for the up and down ramp. Ideally there should only be two values for a given input code, because the voltage could be right at the decision point for a bit, and noise can flip it one or the other way. With two values of equal probability, the standard deviation is 0.5. The program can report all cases for which the standard deviation is higher than a preset limit. After this consistency test, the mean values for the up and down ramp are compared to find whether there is hysteresis in the setup. All cases for which the difference is higher than a preset limit can be reported. Then the data for up and down ramps are combined and the midpoints are calculated; these are basically the levels where the bits change (although it would be nice to have more bits from the DAC for a higher resolution of this measurement).

A linear regression is fitted to this data, reporting slope and intercept. Next the Integral Nonlinearity is calculated, as the sum of squares of the deviation of the measured from the fitted data. The largest positive and negative deviations are reported. Finally, the Differential Nonlinearity is calculated, as the variation in step size. Here an RMS value is reported: root-mean-square value of the cumulative deviations from 1.

Before running this ramp test, the following parameters should be reviewed and adjusted, as necessary: PORT, SETS, SLIMIT and HLIMIT. The parameter VERBOSE controls how much detail is displayed on the screen. Results:

- 10 values with hysteresis > 0.5
- Slope: 1.004048 Intercept: -3.051859
- Integral nonlinearity: 0.06 %
  max pos error: 0.59 @ 1022, max neg error: -0.63 @ 718
- Differential nonlinearity (RMS): 5.589 %
  largest step 1.34 @ 701, smallest step 0.70 @ 275

The value for intercept begs for checking the DAC output with a voltmeter:

static code into DAC	level (mV)	ATmega code
0	0.4	0
1	2.3	0
2	4.2	0
3	6.3	0
4	8.2	1
5	10.2	2
6	12.2	3
10	20.3	7

From this data, the offset is from the ATmega device, not the MAX503. Other devices showed similar results, which leads me to believe that the ATmega has a systematic offset built in, probably due to clock injection.

While running this last test, I found that just connecting a regular voltmeter with standard (not shielded) leads to the net which connects the MAX503 DAC output with the ATmega ADC input brings noise into the measurement, which produces as a significantly higher number of codes with standard deviation >0.5.

6. Sinewave Test with DAC/ADC Combination

The idea of this test is to produce a sinewave shape code sequence. Peaks are chosen at 9 and 1012 lsb, leaving some margin at the extremes for offset. The code is sent to the DAC, which creates a voltage, which in turn is fed back to the ATmega ADC. The digitized values are sent to the PC and analyzed with FFT, to calculate standard measures for ADCs and DACs: THD, SNR, SINAD, ENOB, and SFDR.

The sketch sine.pde is used to control the ATmega device. It makes use of Arduino macros, except in the subroutine which controls the DAC. At the beginning, the file describes also the wiring between the ATmega and the MAX503. An LED+resistor connected to pin 17 serves as pulse indicator. In the setup() section, the values for the sinewave are pre-calculated, because calculating at run time would slow the actual test execution, with possibly different delays for different arguments of the sine function, which would lead to differences in settling time of the analog signals.

To get a maximum number of different codes, I chose the number of samples per period to be prime. Because RAM is limited on the ATmega, 887 was the maximum number which produced correct results. (It is possible to upload a larger sine table as part of the program.) The number of cycles of the sinewave I selected as 100, for the duration of the 887 samples.

The main loop() generates a pulse signal for driving an LED, which turns on or off after every cycle of 887 samples. The ATmega device sends the code which represents the sinewave to the MAX503, alternating with a phase shift of 180 degrees between the two channels, to give the DAC and the ADC front end a good workout. The ATmega device also samples the DAC output, alternating on ports A0 and A1. The measured results from the ADC are output on the serial port to the PC, with encoding described in chapter 2, suitable for LXARDOSCOPE or LXSNDTEST.

When viewing the data with LXARDOSCOPE, the timing must be interpreted as follows:

Click the Timing button. A message is displayed:

***** elapsed time=3.944, 40960 bytes received, 10385 bytes per second, sampling rate= 2689

So in this case the ATmega has sent 2689 samples for each channel, per second. This amounts to 3.03 packets of 887 samples, or 303 cycles of the sinewave. Thus the frequency of the sinewave is actually 303Hz, or the period is 3.3ms. The following screenshot shows the waveforms from the two channels:

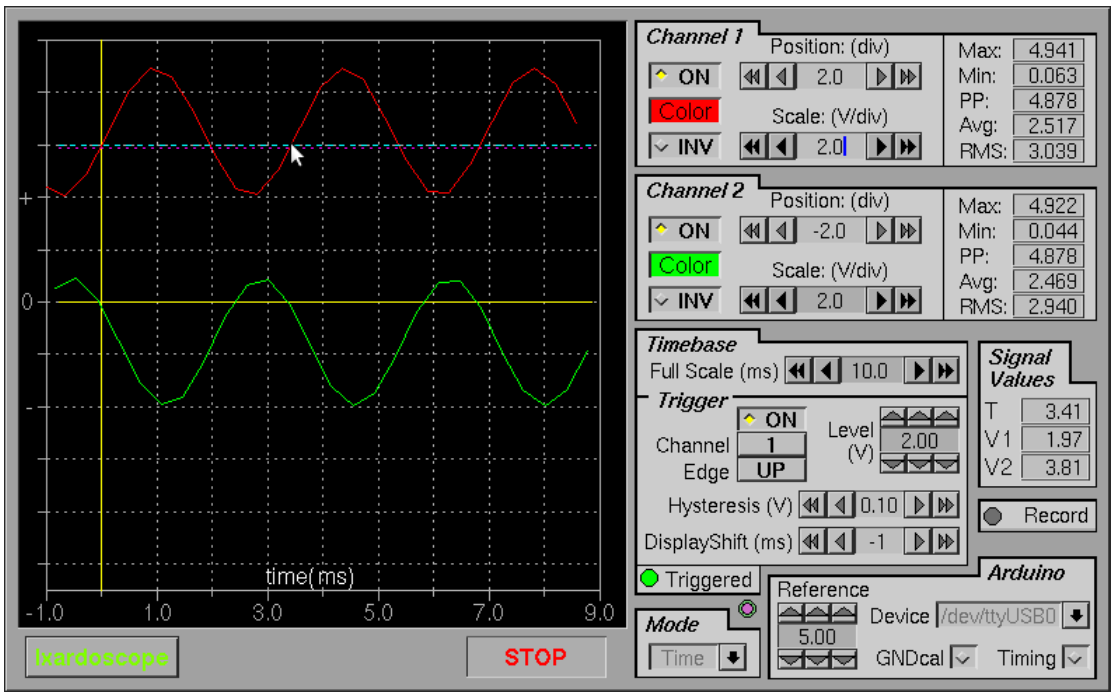

Figure 6: 303Hz sinewave at 2689 samples per second

The time reported for the pointer position is 3.41ms. Obviously the sinewave does not look very nice; the reason is that it is sampled only 9 times per period. To further explore the quality of this sinewave, FFT needs to be used. For LXSNDTEST I recommend the following settings:

- Analysis Mode: Distortion as Analysis Mode
- Signal Source: Sound&TTYdev
- any sound card (it is not used)
- TTYdevice: the port which receives the ATmega data
- Analysis Rate: 887.

Click on START, and a couple of seconds later the spectrum, the noise and the summary are shown. Here are the screenshots:

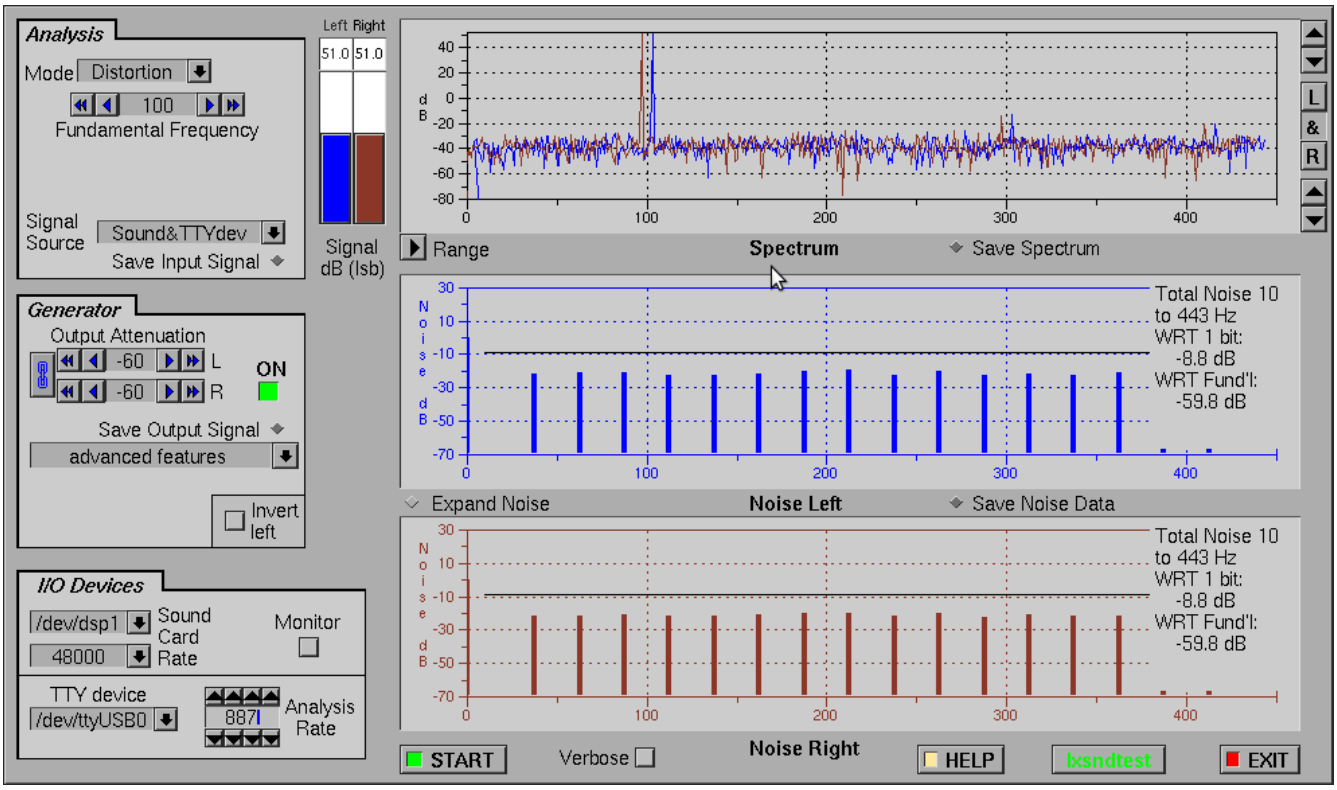

Figure 7: Spectrum from Atmega328 and MAX503 feedback configuration

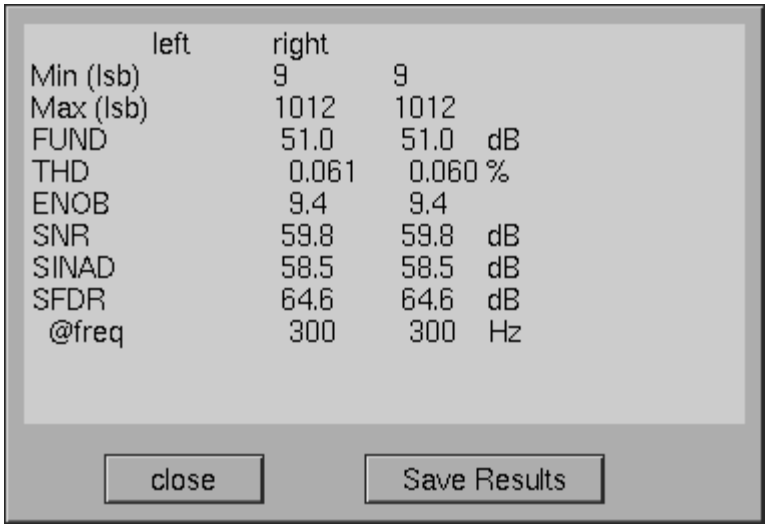

Figure 8: parametric results for ATmega328 and MAX503

I was happy to see an effective number of bits of 9.4, for the DAC and ADC combined, both being 10 bit devices. The setup on the breadboard is certainly not ideal: no ground plane, wires carrying digital signals cross others with analog signals. So I expect somewhat better performance with a better layout (dedicated PC board, with ground planes and shield traces).

7. Reviewing Earlier Results

Feeding the sinewave directly from UCA202 into the ADC, produces the following plot:

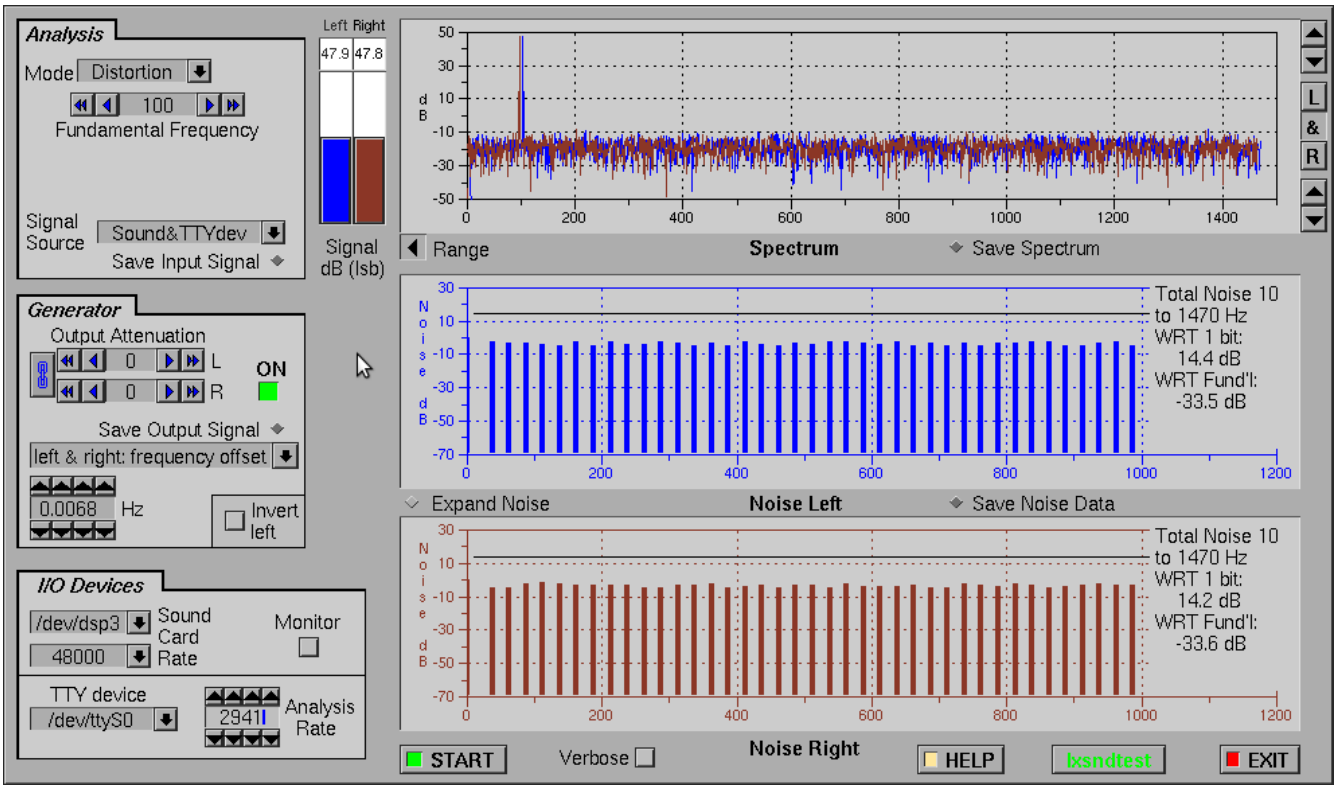

Figure 9: Spectrum for UCA202 driving ATmega328

with the following numerical results:

	left	right	
Min (lsb)	152	156	
Max (lsb)	879	883	
FUND	47.9	47.8	dB
THD	0.131	0.178	%
ENOB	5.3	5.3	! ! ! ! ! ! !
SNR	33.7	33.8	dB
SINAD	33.6	33.7	dB
SFDR	56.4	56.5	dB
@freq	1329	676	Hz

By visual inspection, the noise hovers around -20dB on the spectrum plot, whereas on figure 4 (with input buffer MCP6002), the level is close to -45dB. Granted, the overall level is smaller by 3dB, but this still leaves another 22dB....

Note that the frequency offset for the LXSNDTEST generator had to be changed, due to the difference in clock frequency of the UCA202 and the Extigy soundcards.

Here some more ENOB results for various configurations:

A) UCA DAC > ATmega ADC: 5.3 (as above)

B) UCA DAC > MCP6002 buffer > ATmega ADC: 6.0/6.6 (left/right)

C) UCA DAC > MCP6002 buffer > lowpass 3.3k/18nF > ATmega ADC: 8.6

In these tests, the UCA202 ADC input is connected through a coupling capacitor to the ATmega328 ADC input; the ENOB for this signal path is always 9.5. So the UCA202 input does not report the same level of noise or distortion as the ATmega. I do not have a good explanation why this is happening; my best guess is folding/aliasing of higher frequency signals. However we can conclude that the lowpass filter is beneficial, as already seen in chapter 3, although the effects were less dramatic.

8. Conclusion

Inserting a lowpass filter in front of the ATmega device is beneficial for low noise and distortion.
In the course of the investigation, I have studied several circuits in combination with the Atmega328 and obtained acceptable results:

- preamplifiers for Atmega328 type circuits with oscilloscope standard input impedance
- ATmega328 driving a MAX503 10-bit DAC
- interface circuits for ATmega328 serial port to PC serial port and USB port.
- Electrically insulated interface for ATmega328 serial port.

The program LXARDOSCOPE was useful for a cursory look at the waveforms, to make sure the wiring was correct. The LXSNDTEST program proved to be very useful in these investigations because it can control an ATmega type device and a soundcard for high accuracy measurements combined with FFT processing.

9. Copyright notice

10. Notes

1. This is an updated version with some changes related to aliasing.
2. The schematics were created using the free program GSCHEM. The images were written at 280x960 in png format to preserve details, and then placed with scale of 40% into OpenOffice Writer, combined with the text. The final document was exported to pdf.

July 25, 2012 Oskar Leuthold