From http://www.openmusiclabs.com/learning/digital/atmega-adc/

ATmega ADC

This is an advanced course for those of you who want to push your ATmega ADC to its limits. We will give in-depth information on the inner workings of the ATmega328p ADC (just in case you're using an Arduino), and show you what the trade-offs are for over-clocking or sampling high impedance sources.

But, to start off, we'd like to say: "Don't forget to DIDR!". Perhaps the most overlooked ADC register, the DIDR (Data Input Disable Register) disconnects the digital inputs from which ever ADC channels you are using. This is important for 2 reasons. First off, an analog input will be floating all over the place, and causing the digital input to constantly toggle high and low. This creates excessive noise near the ADC, and burns extra power. Secondly, the digital input and associated DIDR switch have a capacitance associated with them which will slow down your input signal if you're sampling a highly resistive load.

Unfortunately, Arduino doesn't do this for you automatically. But it's as easy as adding the line *DIDR0 = 0×01;* to your *setup()* section to use ADC0. If you're using ADC1 you set bit 1, if ADC2 you set bit 2, and so on. Check page 266 of the ATmega328p datasheet for more information.

So, now that we have that out of the way, how fast *can* the ADC go? The ATmega datasheets give stern warnings about not breaking the speed limit: 50kHz to 200kHz ADC clock for maximum resolution! So how quickly does that resolution fall off with faster speeds? To test this out, we set up an Arduino to sample a pure sine tone with its ADC, and connected a Codec Shield to playback the samples for various ADC clock frequencies. The results are shown below, and an in-depth explanation is given here.

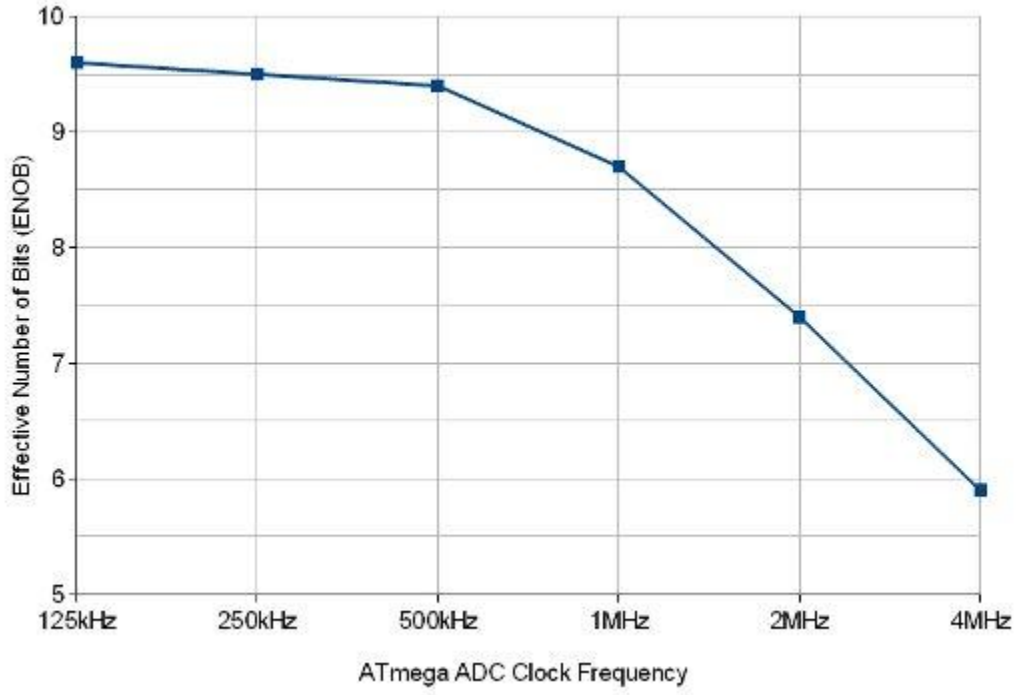

Figure 1 – ATmega ADC resolution versus clock frequency.

The Arduino was clocked at 16MHz, placing a limit on the number of frequencies tested. The fastest frequency (CK/2) refused to work. The next two frequencies down (CK/4 & CK/8) functioned, but gave sporadic audible

clicks in the playback. The remaining settings had no problems at all, with the slower speeds giving less noise and distortion.

It's important to note, that if you want low noise and good frequency resolution from your ADC, you will need to sample at a very consistent rate. To do this, you can use first conversion mode, free running mode, or an interrupt that is a multiple of your ADC clock. All other modes will have 1/2 ADC clock jitter. This will cause your samples to not line-up in time, and slowly wander back and forth, effectively causing a frequency modulation of your signal. To read more about this, check out the in-depth analysis page.

Besides for the clock frequency, the ATmega datasheets also give strict warnings about using source impedances greater than 10kohms. But, why is that the case? As it turns out, it takes a bit of time to charge up the sample and hold capacitor in the ADC input stage. According to the ATmega datasheets, it looks something like this:

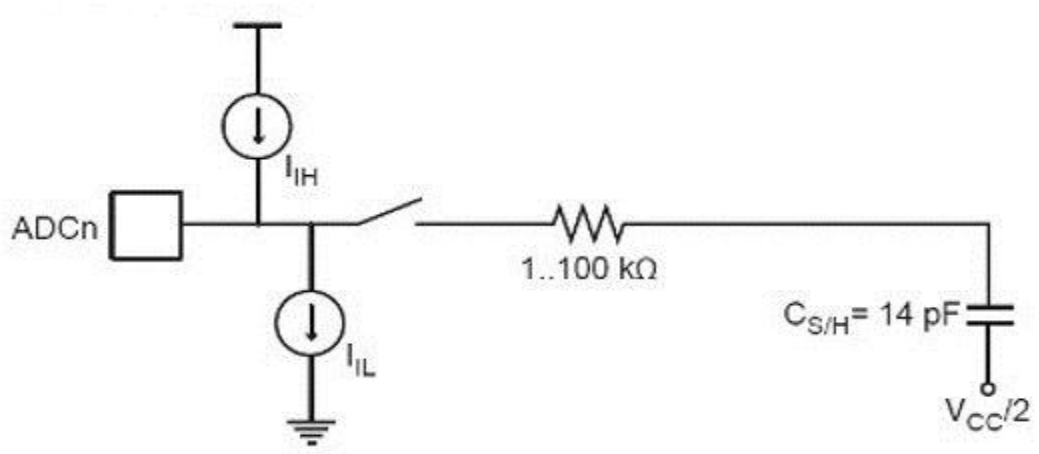

Figure 2 – ATmega ADC input schematic from Atmel datasheet.

They show input leakage current, some input resistance, and 14pF of capacitance. From our tests with 10Mohm resistors, the leakage current was negligible (although this changes with temperature). We also found that the 14pF was actually distributed as 4pF on the ADC MUX, and 10pF on the sample and hold (S/H) capacitor. Therefore, a better approximation for the input stage would be something more like this:

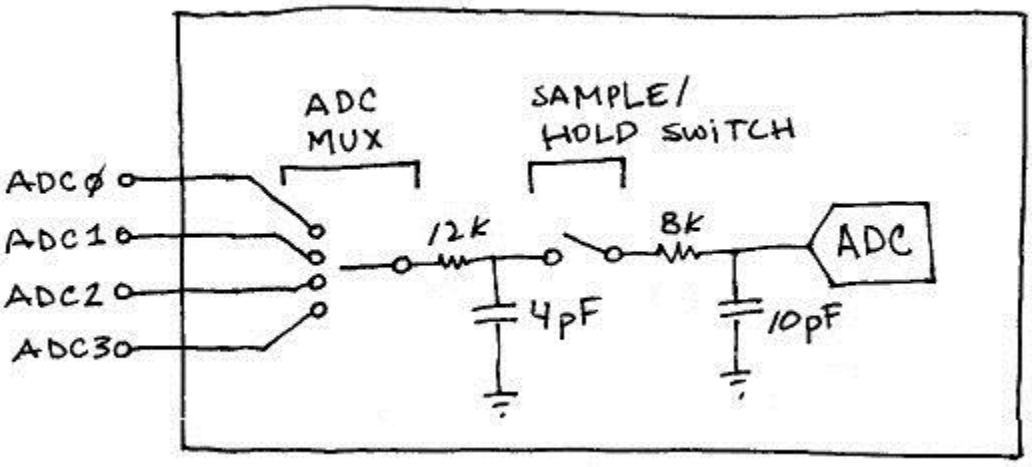

Figure 3 – ATmega ADC internals from experimentation.

So, why does this matter? There are some sensors, like CDS photocells or capacitive touch pads, which have extremely high impedances, and you would normally have to buffer them before sampling with an ADC. But,

you can do without the extra parts if you make sure both the MUX and S/H capacitors are connected to your circuit long enough to transfer their charge and stabilize. The exact order in which these capacitors are connected, and how long they are connected for, varies greatly depending upon which mode you are using the ADC in. The 3 modes we will consider here are first conversion, repeated conversion (single conversion mode), and free running.

With the first conversion, the ADC channel is selected with the MUX, the ADC is enabled, and sampling begins. Since it is a first sample, it takes a total of 25 ADC clock cycles to complete. The MUX capacitor is engaged immediately upon changing ADMUX, and the S/H capacitor is engaged as soon as the ADC is enabled. If the ADC is enabled at the same time sampling is initiated, the S/H capacitor is engaged after a 1/2 clock cycle delay. It is then held on for 1.5 clock cycles, and again for 1 clock cycle after the initialization period. Once the sample is complete, the S/H capacitor is re-engaged, and left that way until the ADC is turned off. The full timing diagram is shown below.

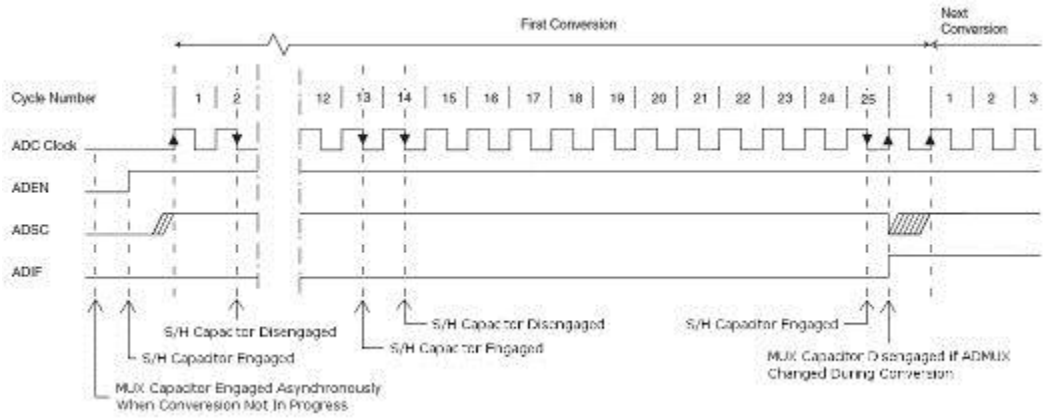

Figure 4 – ATmega ADC first conversion timing diagram (annotated from datasheet – click for larger image).

For single conversion mode, the MUX capacitor is again engaged upon changing ADMUX, but the S/H capacitor is always left connected, except during the actual conversion period. As shown in the timing diagram below, this gives a much longer duration for charge to transfer, making it preferable for high impedance sources.

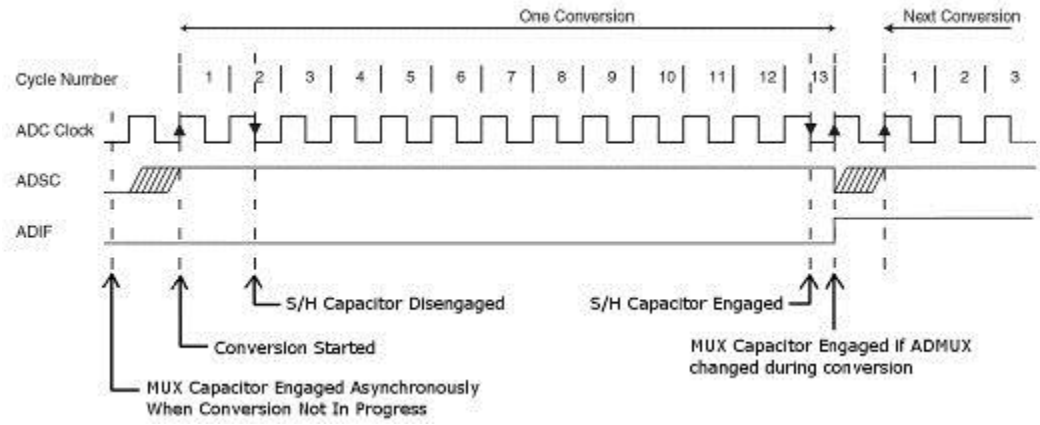

Figure 5 – ATmega ADC single conversion timing diagram (annotated from datasheet – click for larger image).

For free running mode, the ADC behaves similarly to single conversion mode, except that the MUX changes directly before sampling, so there is very little time for charge transfer. Its timing diagram is shown below.

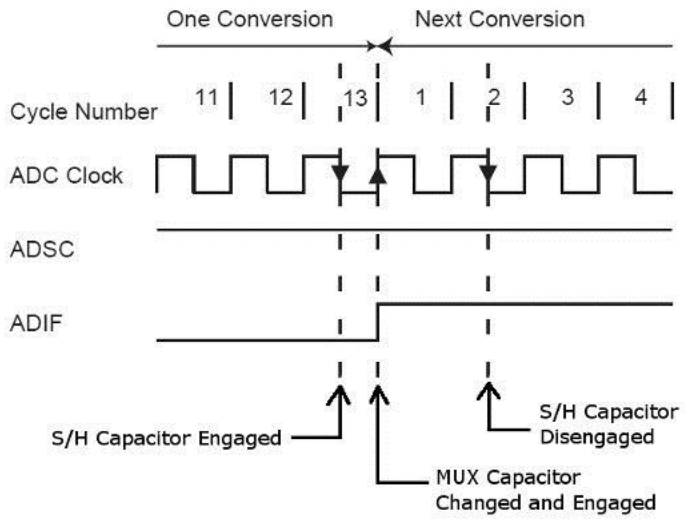

Figure 6 – ATmega ADC free running mode timing diagram (annotated from datasheet).

For any of these modes, you want to be sure that the S/H capacitor is engaged long enough for your signal to propagate. This is usually 4*R*C, where R is your source impedance, and C is the sum of all capacitances (MUX, S/H, PCB traces, etc.). For example, if you have a 10Mohm CDS photocell, and 6pF of PCB trace capacitance plus 14pF of ADC capacitance, you will have to wait 4*10Mohm*20pF = 800us. One way to do that is to set ADMUX to your channel, enable the ADC, wait 800us, and then tell the ADC to sample. Another way is to place a large (~1nF) capacitor at the ADC input, so that the charge is taken from the capacitor and not the resistor, giving minimal voltage drop. This gives a slower frequency response, but doesn't require the wait cycle. In either case, you can sample as high of an impedance as you like, as long as you wait long enough.

Use in Audio Applications

There doesn't seem to be much loss for increasing the ADC clock frequency up to 500kHz. Using free running mode, this gives a sample rate of 38kHz, which is really good for most low-fi applications. But, Considering the difficulty in making good anti-aliasing filters, it might be worth going up to 76kHz. Frequencies above that should be avoided, as the ADC creates sporadic clicks.

if you intend to process the data on-board and use the PWM to create a corresponding output, it's very important to match the ADC and PWM sample rates. The best way to do this is to interrupt on the PWM overflow condition, and initiate a new sample at that time. Since the ADC and PWM are clocked from the same CPU clock, you shouldn't have to worry about clock jitter, as long as the PWM sample rate is less than the ADC clock divided by 14.