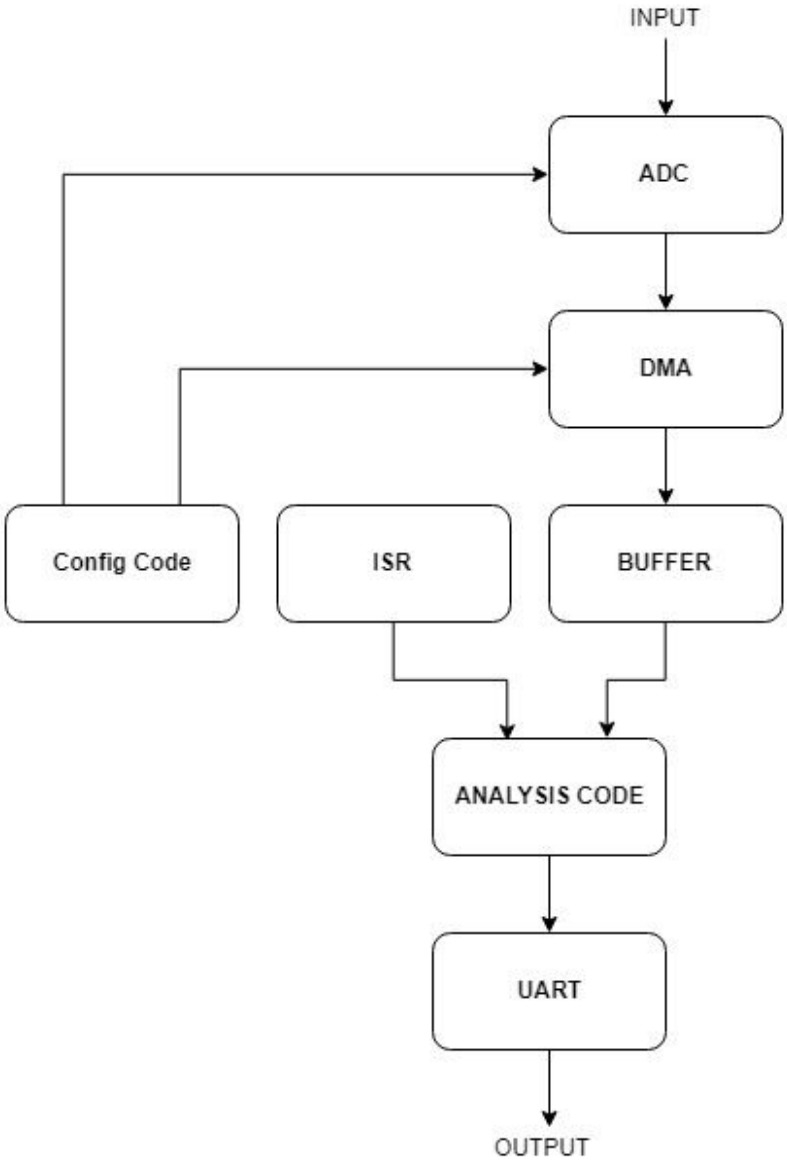
PROJECT 3: ADC AND DMA

D. Doty
ECEN 5813
Prof. K. Gross
University of Colorado Boulder
Fall 2018

Part 1:

Block Diagram:



Part 2:

1. What is the behavior of the system when no input signal is applied?

The system wiggles around randomly due to whatever noise it picks up from EMI. Mostly floats near the mid to upper end of the sample range.

Part 3:

1. What source and destination DMA read and write size did you choose and why?

I chose to use 16 bit transfers. Originally I opted to use 32bit transfers because I thought it would make it faster, but eventually I realized the 32 bits were causing me a lot of heartache when converting them to unsigned numbers in the calculations. I decided to change to 16 bit buffers because the sign conversion is much more straightforward that way. So far, the system

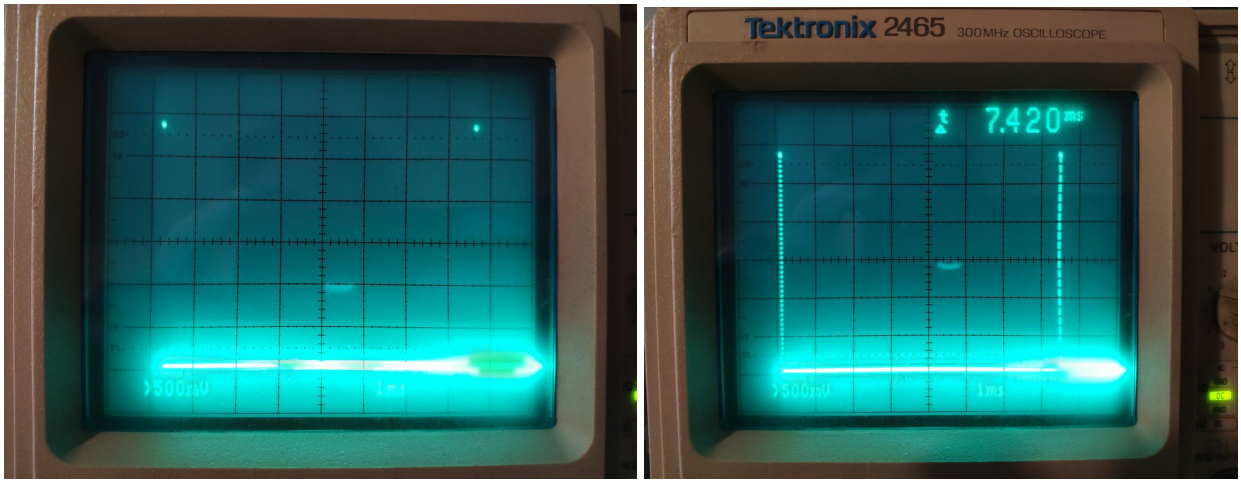
seems to stay well ahead of the DMA controller and it doesn't seem to be a problem. The ideal solution would be to use a single ended input and do an unsigned conversion with a 32 bit buffer as that is fastest through the machine and requires no extra conversion.

Part 4:

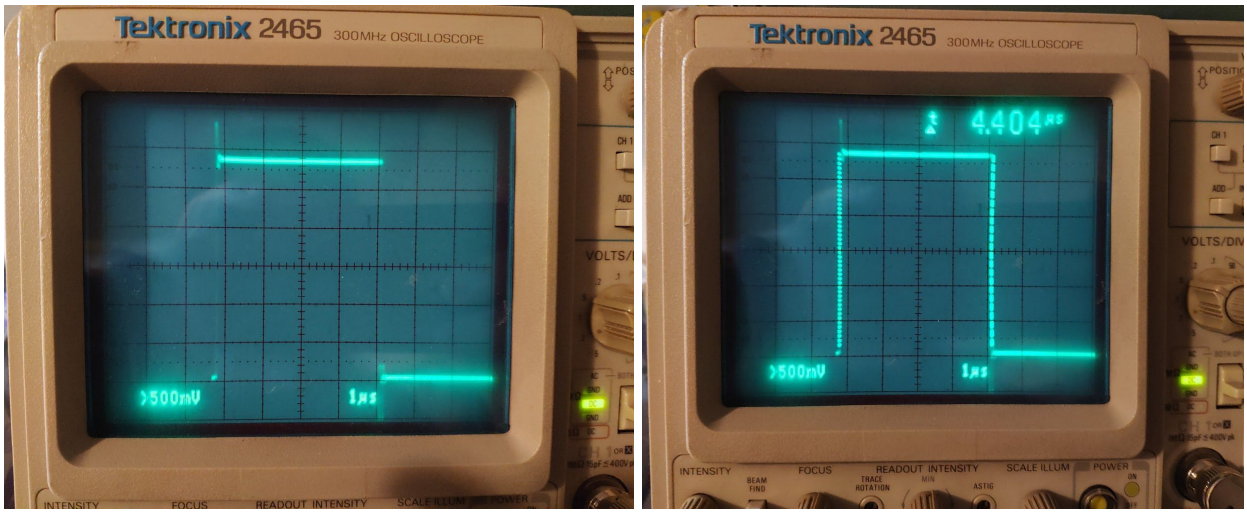
1. What is the consequence of a lengthy or delayed interrupt service for the data retrieved by the DMA controller?

ADC samples are missed if the DMA is disabled for too long, since it converts continuously whether the DMA is ready to grab the result or not. I implemented the double buffer DMA solution so I don't believe I ever miss a sample.

Oscilloscope Screenshots:



Interrupt to Interrupt timing, left image without cursors, right image with cursors. Sample rate was ~8ks/s into a 64 sample buffer, which should produce interrupts at ~8mS, very close to measured 7.420mS.



Single interrupt, left image without cursors, right image with cursors. ISR runs in ~4.4uS. Full size images are in /doc folder of github.

Part 5:

1. What effect does changing the buffer size have on system processing efficiency?

A larger buffer increases the efficiency of the analysis part of the application because the processor is able to work through a longer set of sequential data. This increases the number of cache hits, and through loop unrolling possibly eliminates some of the loop condition checks. In a pipelined processor this also increases performance because the pipeline does not need to be flushed as often.

2. What other effect(s) does changing the buffer size have on the system?

Increasing the buffer size would increase the latency of the system because that would mean a longer time taking samples, a longer time analyzing them, and therefore a longer time before they're reported. If this is acceptable though, it could also mean more time for a secondary task, allowing it to take advantage of the efficiency gains mentioned in question 1.

Extras:

I ended up using the double buffer solution originally proposed in the assignment, with 2 64 entry buffers. At first I attempted to set up 2x DMA channels with one pointing to each half of the buffer so I could simply disable one channel and enable the other in the ISR, but it appears that only DMA channel 0 supports the ADC. I didn't need to use the modulo option at this point since I have to manually change the destination address in every ISR to the other half of the buffer.

I implemented a look up table with interpolation for dBFS calculations and those are reported in the terminal.

I also implemented a short extra function that prints a graphical representation of the ADC signal in the terminal. This can be enabled or disabled with the flag PRINT_PRETTY_LINES defined at the top of main. It works best with only one of line or text display activated at a time, as printing both significantly slows down the execution.