Simple DSP Framework for Arduino Duo

Barry L. Dorr, P.E.
Department of Electrical and Computer Engineering
San Diego State University

January 18, 2021

Introduction

Analog inputs and outputs must be sampled at a fixed sample rate when used in a system that uses signal processing such as a control loop or a digital filter. When using the Arduino, it is convenient to use the delay() function to establish a fixed amount of time between samples and the analogRead() or analogWrite() functions to control the ADC and DAC. The problem with this approach is that during the delay period and during analog reads or writes, the processor is not available for other tasks. As a result, the processor may be able to read and/or write the ADC/DAC at the desired sample rate, but it will be excessively loaded and therefore not available to do much else.

This note describes a Simple DSP Framework implemented as an Arduino sketch that eliminates the problems noted above and allows the Arduino Due to be used for simple DSP functions such as control loops or digital filters.

A Better Way to Maintain a Fixed Sample Rate

Ideally, reading an ADC or writing to a DAC should happen in the background so the microprocessor's main loop is free for other tasks. This is done with the strategy below:

1) Configure the hardware timer to interrupt the processor at the desired sample rate.
2) Create an Interrupt Service Routine (ISR) that does the following in the order shown below:
    a. Read the result of the last ADC conversion to a global variable AdcResult.
    b. Command the ADC to begin another conversion.
    c. Write the last computed DAC result (global variable DacSample) to the DAC.
    d. Command the DAC to begin conversion.
    e. Iterate the digital filter or control loop such that it accepts AdcResult and generates the next DacSample.
    f. Exit the interrupt.

Figure 1 shows the sequence of events resulting from this strategy. The increased efficiency comes from the fact that the microprocessor never has to wait for the delay, ADC, or DAC, and the loop() function is available to do background tasks.

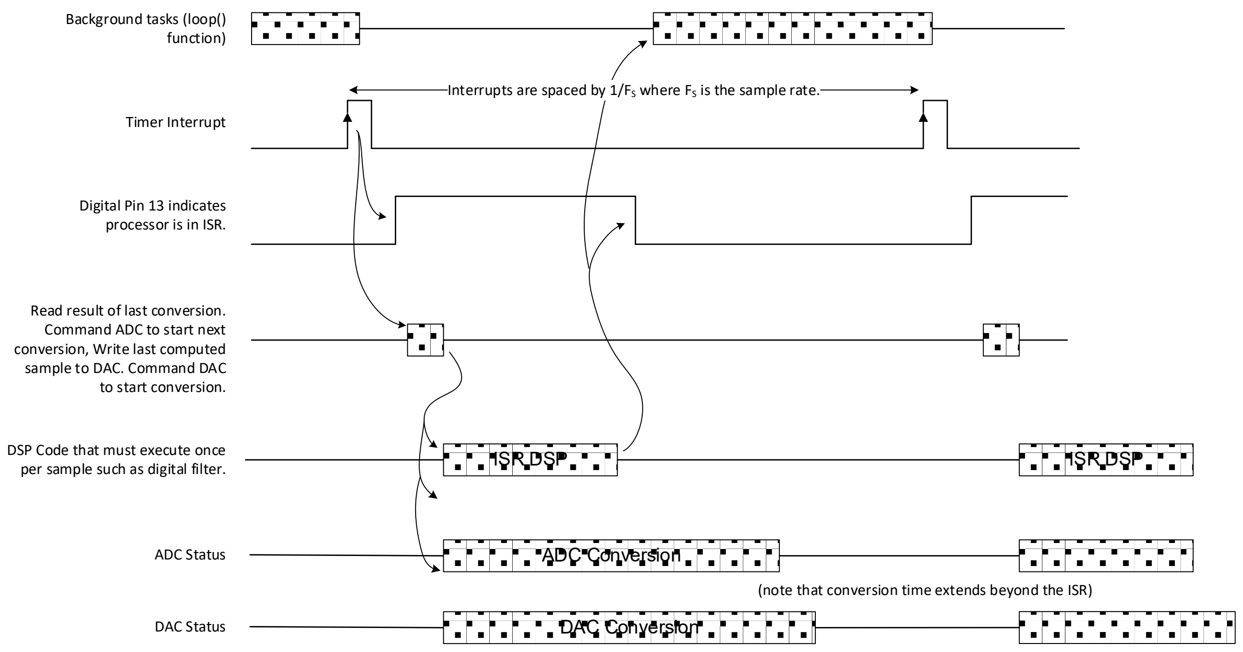
Background tasks (loop() function)
Interrupts are spaced by 1/Fs where Fs is the sample rate.
Timer Interrupt
Digital Pin 13 indicates processor is in ISR.
Read result of last conversion. Command ADC to start next conversion, Write last computed sample to DAC. Command DAC to start conversion.
DSP Code that must execute once per sample such as digital filter.
ISR DSP
ISR DSP
ADC Status
ADC Conversion
(note that conversion time extends beyond the ISR)
DAC Status
DAC Conversion

Figure 1 - Timing Sequence for periodic sampling

The Simple Framework

The Arduino Due was chosen as the target platform because it has a 12-bit ADC, 12-bit DAC and an 84 MHz processor, so it can effectively be used for control loops or audio processing.

The framework can be found here. The digital sampling rate is 44.1 kHz, but it can be changed with compiler constant FS. There are three simple digital lowpass filters in the Interrupt Service Routine so you can experiment with them. All three are described in detail in the Digital Filtering chapter of "Ten Essential Skills for Electrical Engineers," Wiley 2014. Two are very efficient fixed point filters, and the other is a floating point filter. Just uncomment the one you wish to use.

Functions of Arduino Due Pins

Pin	Function	Notes
Analog Input A0	Accepts analog input to the ADC	The range of the ADC set for 0 – 3.3 V. If you input a sinewave, make sure to offset it by 1.65 V so it stays within the range of the ADC.
Analog Output DAC1	DAC output signal	The range of the DAC output signal is 0 to 3.3 V
Digital Pin 13 (LED Pin)	Provides indication that processor is in the interrupt routine. Also used to show the	This pin is set high when the ISR is entered and returned low at the end of the ISR. By comparing its 'high' time to the sample period, you can estimate your processor loading.

	processor is not overloaded.	If the time between rising edges of this signal is exactly the desired sample rate, then your processor is not overloaded.

Performance

Here are some performance results for the Arduino Duo at a sample rate of 44.1 kHz. These were measured using Digital Pin 13 as described above.

ISR Operation	ISR Execution Time	Processor Usage[1]	Notes
Write ADC value to DAC	2.56 us	11.29%	Simple function demonstrates sampling at a constant sample rate.
Filter ADC input with FIR/IIR lowpass filter and write filter output to DAC – integer math	2.99 us	13.19%	This filter is extremely efficient and provides the response of an RC lowpass filter.
Filter ADC input with averaging lowpass filter and write filter output to DAC– integer math	3.47 us	15.30%	The averaging lowpass filter has nulls in its frequency response and can be used to reject tones. Integer math and a circular buffer make it very efficient.
Filter ADC input with FIR/IIR lowpass filter and write filter output to DAC – floating-point math	12.86 us	56.71%	This filter performs the same function as the FIR/IIR Lowpass filter shown above, but floating point emulation in the ARM increases the processor load.

Conclusion

The Arduino Due can be used to capture ADC samples and write DAC samples at a fixed sample rate while maintaining ample processing power for simple DSP structures such as small digital filters or control loops as well as background processes. However, sampling should be done as shown in this note – not with the delay(), analogRead() or analogWrite() functions. The sample code in the Simple DSP Framework shows how to set up the timer, interrupts, ADC, and DAC. It also provides several filtering examples that can be used to benchmark your system performance or include in your project. If you plan to use floating point code in your system, note that it will load your processor as shown in the table above. If you want to learn more about simple digital filters, see the Digital filtering chapter in this book.

[1] Note that the overhead time required for entering and exiting the ISR are ignored. As a result, the estimates in this column are slightly low.