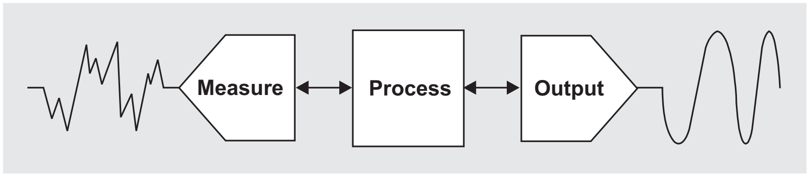
Streamlining the mixed-signal path with the signal-chain-on-chip MSP430F169

By Zack Albus (Email: j-albus@ti.com)
MSP430 Applications

With few exceptions, electronic measurement and control system designs incorporate varying degrees of three distinct functions: measurement of the environment, processing of data, and output back into the environment. This process can be anything from the measurement of thermistor resistance and display of temperature on an LCD to digital filtering of an analog signal. The common thread among many processes is that they often perform some level of digital conversion of an analog signal, computation and decision-making in the digital world, and conversion back into the analog domain. Figure 1 illustrates these elements that make up the analog signal chain.

Figure 1. Typical measurement and control system



Over the past decade, advancements in semiconductor integration have facilitated more robust combinations of digital processing and analog peripherals. While such devices are nothing new to today's standards, there are a number of trade-offs that must be evaluated to choose the level of integration most suitable for a given application. Key issues include system performance, size, and cost. As system complexity increases, increased integration can provide a smaller and lower-cost design capable of performance equal to or even better than that of the discrete alternative. Take, for instance, a typical discrete analog signal chain solution as compared to an integrated signal chain solution, both designed to solve the same monitor and control system function. An example for each system is shown in Figure 2.

In both systems, a variable resistance generates a voltage level that is sampled by the ADC. The conversion result is processed and used to determine the update rate of the DAC and, consequently, the analog output signal frequency. The output signal itself is a 12-bit sine wave and is made up of 16 steps or data points. While the analysis and results are based on the specific software implemented for each case, the same approach presented here can be used to determine CPU performance across any application.

Figure 2. Discrete system (top) vs. the integrated MSP430F169

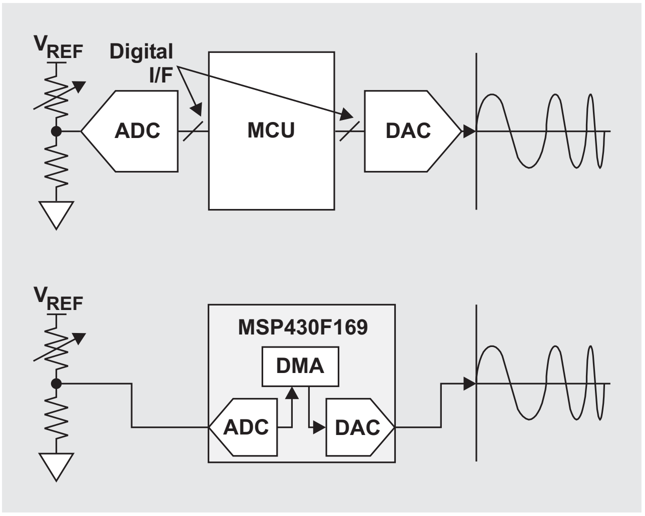
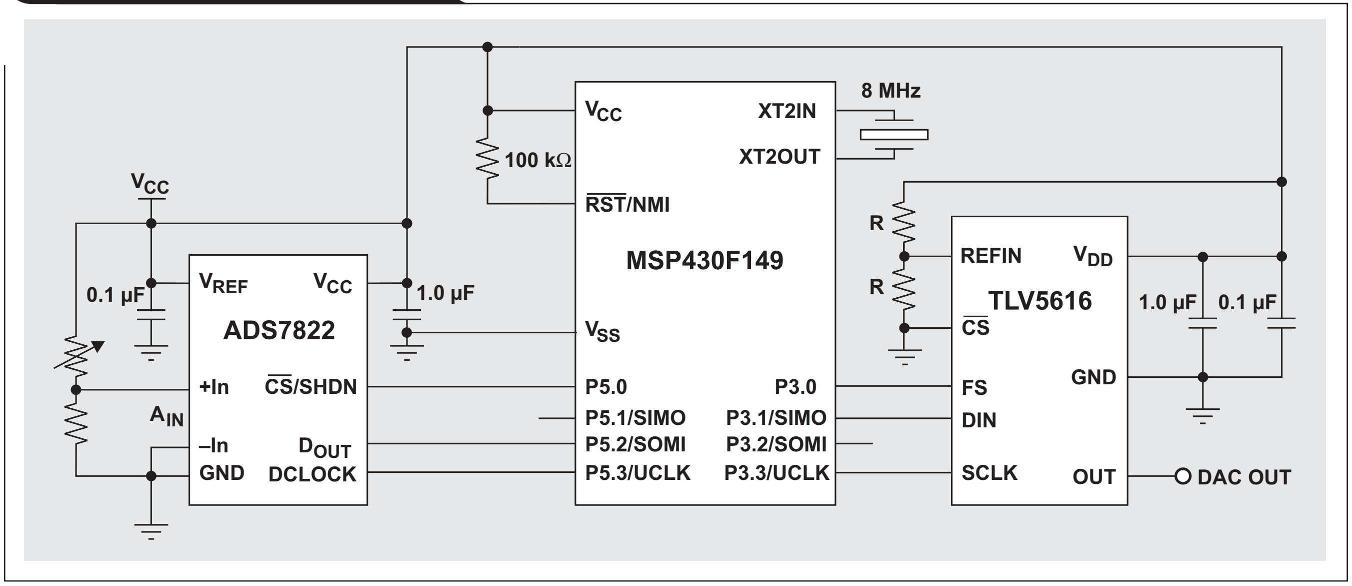


Figure 3. Discrete analog system

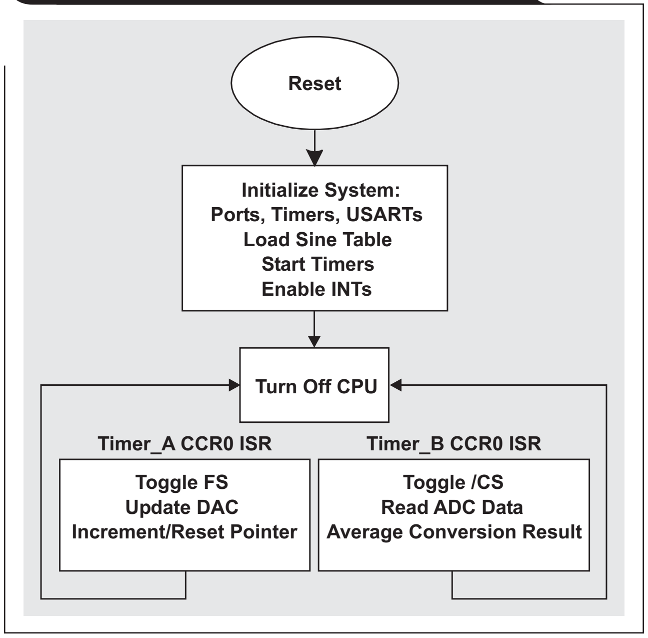


Case 1: Discrete peripheral system

The discrete signal chain solution is envisioned with the MSP430F149, an external ADC, an external DAC, and a serial interface to connect the system. The ADC measures and converts the variable voltage, and the result is used to process and determine the output frequency of a sine wave generated by the DAC. This discrete solution is detailed in Figure 3.

As shown, the external ADC and DAC are controlled via separate SPI interfaces of the microcontroller. Clocking of the MCU is performed at a maximum clock speed of 8 MHz and is input at the XT2 terminals. The external resonator clock source drives two 16-bit timers that provide individually controllable interrupt durations that initiate communication with each of the external peripherals. Sampling of the ADC occurs at a rate of 8 kSPS as set by Timer_B, and data is transferred from the ADC at a 1-MHz serial clock frequency via the hardware USART. Data loading of the external DAC occurs at a variable frequency as determined by Timer_A. The frequency at which Timer_A updates the DAC is determined by an 8-sample average of the ADC conversion result. Using a 16-point sine data table, this configuration provides an adjustable DAC-generated sine wave. Figure 4 shows the software flow for the discrete component system and represents common operation of an MCU using interrupt processing with multiple peripherals.

Figure 4. Discrete system software flow



To characterize the performance of the system, a loading analysis of the CPU is performed. Time spent by the CPU servicing the ADC during execution of the Timer_B interrupt service routine (ISR) requires a minimum of 60 CPU clock cycles (60 MCLKs). This includes toggling of the ADC /CS, receiving 16 bits of data, and averaging 8 conversion samples. The serial data transfer from ADC to MCU takes additional time during which the CPU must wait for the completion of each data byte transfer. The number of bits to be transferred and the ADC serial clock rate used determine this additional time.

The ISR executes in 60 MCLKs and requires an additional handling time of 11 MCLKs (the number of cycles needed to enter and exit each ISR). Every 8th ADC sample, the average result is calculated; and Timer_B CCR0 is updated, triggering the DAC ISR. The average and update operation adds 29 MCLKs to the ADC ISR every 8th time through the routine.

Maximum Timer_B ISR execution time (every 8th sample):

$$\frac{(60+29+11)}{8\text{ MHz}}+\frac{16}{1\text{ MHz}}=28.5\ \mu\text{s}$$

Average Timer_B ISR execution time (average time required to receive one conversion from the ADC):

$$\frac{(60+11)\times 87.5\% + (60+29+11)\times 12.5\%}{8\text{ MHz}} + \frac{16}{1\text{ MHz}} = 25.3\ \mu\text{s}$$

CPU loading required to complete the total ISR execution time can be estimated in terms of the average MCLK cycles required to service the ADC.

Timer_B ISR average MCLK cycles required:

$$25.3\ \mu\text{s} \times 8\text{ MHz} = 202.4\text{ MCLKs}$$

Given the 8-kSPS ADC sample rate, the CPU must enter the Timer_B ISR 8000 times per second, for a total of 1,619,200 MCLKs each second—a CPU loading of ~1.62 MIPS required to communicate with the ADC and handle data. This represents approximately 20% of the CPU's total available instruction execution time (8 MIPS maximum), leaving 6,380,800 CPU cycles to perform other MCU functions such as data transfer to the external DAC.

As mentioned earlier, the DAC is updated at a variable rate depending on the time between interrupts as determined by Timer_A. The DAC update interrupt interval is proportional to the value in Timer_A CCR0. The ISR requires a minimum of 51 MCLKs to complete, including toggling /FS, transferring 16 bits to the DAC, and servicing the sine table pointer.

Since the DAC cannot be updated faster than the Timer_A ISR can be executed, the maximum update rate, and consequently the maximum DAC output frequency achievable, is calculated based on the ISR time required to update the DAC.

In contrast to the time spent executing the ISR servicing the ADC, the CPU does not have to wait for all 16 data bits to be transferred to the DAC before exiting the ISR. After writing the least significant byte to the SPI transmit buffer, the hardware USART module handles transmission of the data to the DAC, allowing the CPU to continue instruction execution. The actual time spent updating the DAC is a combination of the time spent executing the ISR and the time required to complete data transmission to the external DAC. A complete DAC update cycle is shown in Figure 5.

To avoid any possible transmit data overrun, the SPI transmit buffer is tested after the DAC ISR is entered. If data is still being transmitted when the ISR is entered, the CPU will wait until the transfer is complete before sending the next sync pulse to the external DAC.

In addition to the 51 MCLKs to execute the ISR, an additional 4 clocks are required every 16th loop through the routine, which resets the sine table pointer to the first value in the array. This causes every 16th DAC update (or 6.25% of the DAC updates) to require a small additional execution time, which is taken into account when actual CPU loading is calculated. Also included are the 11 cycles required to enter and exit the ISR.

Average Timer_A ISR execution time:

$$\frac{(51+11)\times 93.75\% + (51+4+11)\times 6.25\%}{8\text{ MHz}} \cong 7.78\ \mu\text{s} \approx 62.2\text{ MCLKs}$$

Timer_A ISR CPU loading for a given f_{OUT}:

$$62.2\text{ MCLKs} \times 16\text{ points} \times f_{OUT} = \text{MIPS}$$

The maximum DAC frequency based on the measured time to transmit each 16-bit data word is given by:

$$\frac{8\text{ MHz}}{62.2\text{ MCLKs} \times 16\text{ points}} \cong 8\text{ kHz}$$

The maximum frequency represents a 100% loading of the CPU. In reality this is not feasible, as CPU time must also be spent servicing the ADC. The maximum frequency achievable by the DAC is more correctly established when the required ISR execution time needed to sample the external ADC is taken into account. This is a critical requirement if distortion cannot be tolerated in the given application. Figure 6 shows the effect on the DAC output as the update frequency increases beyond the speed of the Timer_B ISR sampling the ADC.

Figure 5. External DAC update (Timer_A ISR) cycle time

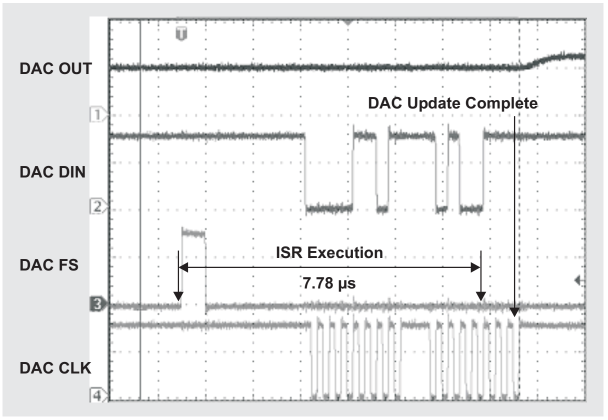
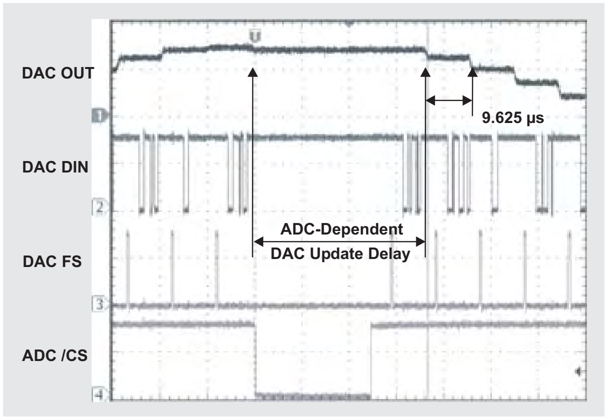


Figure 6. Sine distortion due to ADC ISR service delay



The desired DAC update frequency, or DAC ISR entry frequency, is established by the value in Timer_A CCR0, which for this example is 0x004C = 76 counts. The actual time between DAC ISR entries is 77 counts and includes one additional timer count occurring when the timer rolls from 76 to 0. For the example Timer_A CCR0 value with an 8-MHz Timer_A clock, the expected time delay between DAC updates is 9.625 µs as shown in Figure 6. However, when the ADC is sampled, a much longer delay is incurred; and the shape of the desired sine wave is affected. This effect can be eliminated by establishing a maximum update rate of the DAC and can be estimated by the total time required to execute each ISR. Keep in mind that this is the maximum time, not the average. The total ADC and DAC transfer time is 36.75 µs, providing a maximum DAC output frequency of ~1.7 kHz. The degree to which distortion can be allowed in the DAC output and the methods used to reduce it are application-specific and will determine the maximum acceptable output frequency.

Each complete sine output cycle at this frequency requires 1700 SPS × 16 data points to be transferred to the external DAC each second. With 62.2 MCLKs required to complete each DAC update, the CPU loading due to the DAC ISR at the maximum f_{OUT} is given by:

$$1.7\text{ kHz} \times 16\text{ points} \times 62.2\text{ MCLKs} \cong 1.7\text{ MIPS}$$

This represents a maximum loading of approximately 21% of the total available instruction throughput of the CPU. As the DAC update frequency is reduced, the CPU loading percentage will decrease accordingly. However, when CPU bandwidth is determined for any application, the maximum potential loading must be taken into account to ensure desired system performance over the entire operational range of the system.

Total CPU loading to service both the external ADC and DAC comes to slightly over 3.32 MIPS, leaving 58% of the CPU's available time to perform additional tasks. With the exception of calculating the ADC average and servicing the sine table pointer, the CPU spends this time doing nothing more than transferring data between the external components of the system. To eliminate the impact of data handling on the CPU, the communication between each element of the system must be streamlined. Integration of the analog functions in the system with the MCU is the step required to achieve this efficiency.

Case 2: Integrated peripheral system

By utilizing the highly integrated analog and digital functionality of the MSP430F169, the system described in the previous section is realized completely with a single-chip silicon solution. Figure 7 shows the integrated MCU system.

Integration of the ADC and DAC functions on-chip with the MCU greatly simplifies the system design in comparison to the discrete case. The serial communication protocols to the ADC and DAC have been removed from the CPU and off-loaded to the ADC and DAC internal modules. The ADC conversion averaging function is still performed by the CPU; but, in the case of the DAC, data transfer and pointer handling are entirely performed by the on-chip DMA module. DAC output frequency adjustment is made by interrupting the DMA instead of the CPU, freeing up resources for other tasks. The software flow for the integrated approach is shown in Figure 8.

The integrated solution uses Timer_B to establish the on-chip ADC12 sample/convert trigger for an ADC sample rate of 8 kSPS. This is given by Timer_B CCR0, which is set to 0x03E7 or 999. Timer_B is clocked with SMCLK = 8 MHz as in the external peripheral case. The ADC12 module is configured to perform a repeat conversion on a single channel: A0. Every 1000 Timer_B counts, or 0.125 ms, the ADC12 is triggered and performs a sample/convert of A0,

Figure 7. Integrated analog system

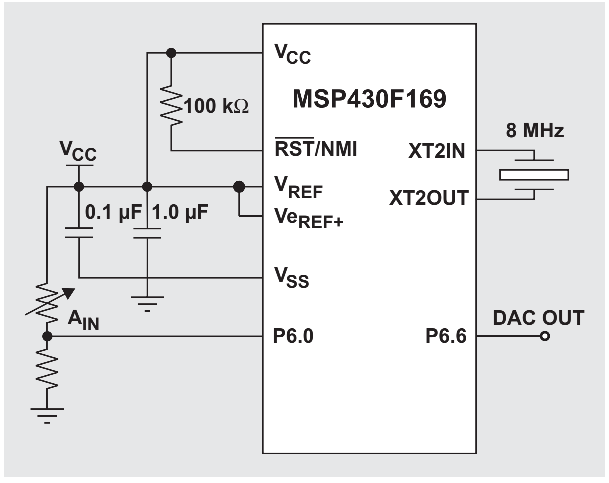
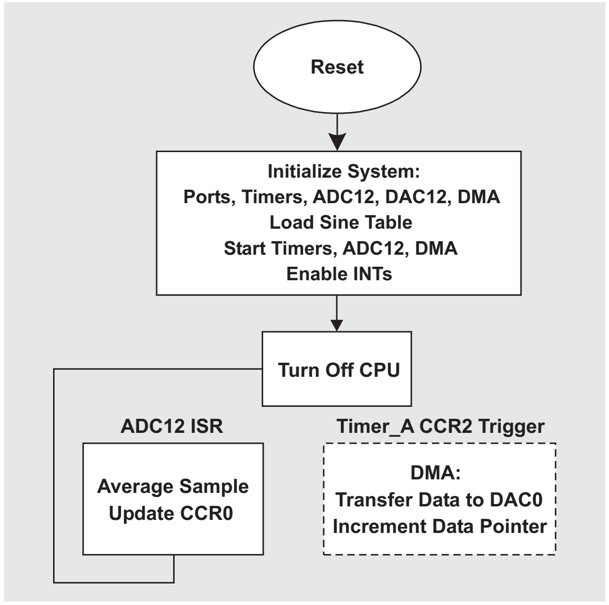


Figure 8. Integrated system software flow



stores the conversion result in the ADC12MEM0 register, and generates an interrupt.

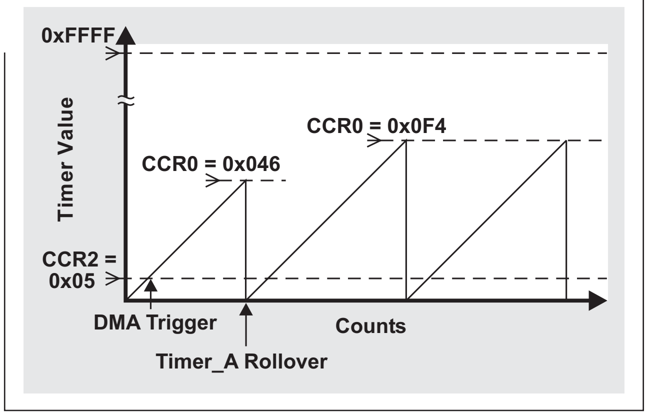
The ADC12 ISR requires a maximum of 50 MCLKs to complete. This is for every 8th interrupt, which includes the averaging of the conversion result. (ISR execution durations for samples 1 through 7 require only 21 MCLKs.) The averaged result is then moved to Timer_A CCR0, which defines the timer count between triggers to the DMA loading data to channel 0 of the DAC12 module. The total MIPS required to perform the ADC conversion handling and averaging is given in the following equation.

On-chip ADC ISR MCLKs per second:

$$\frac{(21+11)\times 7000+(50+11)\times 1000}{1,000,000} \approx 0.29 \text{ MIPS}$$

Total CPU loading for the ADC12 ISR is 3.6% of the total available CPU—a reduction of greater than five times the CPU loading as compared to the external ADC scenario. While this reduction is impressive, more compelling is the increased performance achieved by bringing the DAC on-chip. Along with the DMA available on the MSP430F169, DAC updating can be performed completely transparent to the CPU.

Figure 9. Timer_A operation



The DAC12 module is configured in 12-bit, unsigned binary mode. As defined earlier, Timer_A is clocked by SMCLK and is set to count up to CCR0 as determined by the ADC12 conversion result average. The DMA trigger moving data from the sine table to the DAC is issued when the value in Timer_A CCR2 is reached by Timer_A. Timer_A CCR2 need not be changed and can range in value from 0 to Timer_A CCR0. When Timer_A crosses the value in Timer_A CCR2, a trigger is sent to DMA channel 0. Since Timer_A CCR0 will be continuously modified to change the DAC update rate, Timer_A CCR2 should be assigned a value less than the smallest value to be written to Timer_A CCR0. The concept of updating the DAC via the DMA is shown in Figure 9.

Channel 0 of the DMA is configured to transfer data from RAM to DAC12_0 upon reception of a Timer_A CCR2 trigger. The source address for the DMA is the first location of the sine table data and is auto-incremented each time a value is moved to DAC12_0. Once all 16 data points are sent to the DAC, the DMA source address resets to the first value of the sine table and repeats. The number of data values transferred by the DMA before address reset is defined in DMA0SZ and is equal to the length of the sine data table.

Triggering of the DMA and the transfer of sine data to the DAC12 module is performed entirely transparent to the CPU. Although the CPU executes no instructions, each DMA transfer does require 2 MCLKs to complete. One MCLK cycle is required to retrieve the source data, and the second moves the data to the destination location. During this time, the CPU is halted for 2 cycles and the DMA transfers the required data. Effective CPU loading can be determined based on the required 2 MCLKs per DMA transfer.

For the same 1.7-kHz maximum DAC f_{OUT} in the external ADC/DAC example, a total of 54,400 MCLKs are required to perform an update cycle through the complete sine table via the DMA.

$$\frac{1.7 \text{ kHz}\times 2\times 16}{1,000,000} = 0.055 \text{ MIPS}$$

This is 0.68% of the total available CPU throughput. Totaled with the time spent calculating the ADC12 conversion result average, the required total CPU loading for the system operating with a DAC output frequency of 1.7 kHz is 0.35 MIPS. This represents a factor of almost 10 times fewer instructions per second than that for the exact same system using external peripherals.

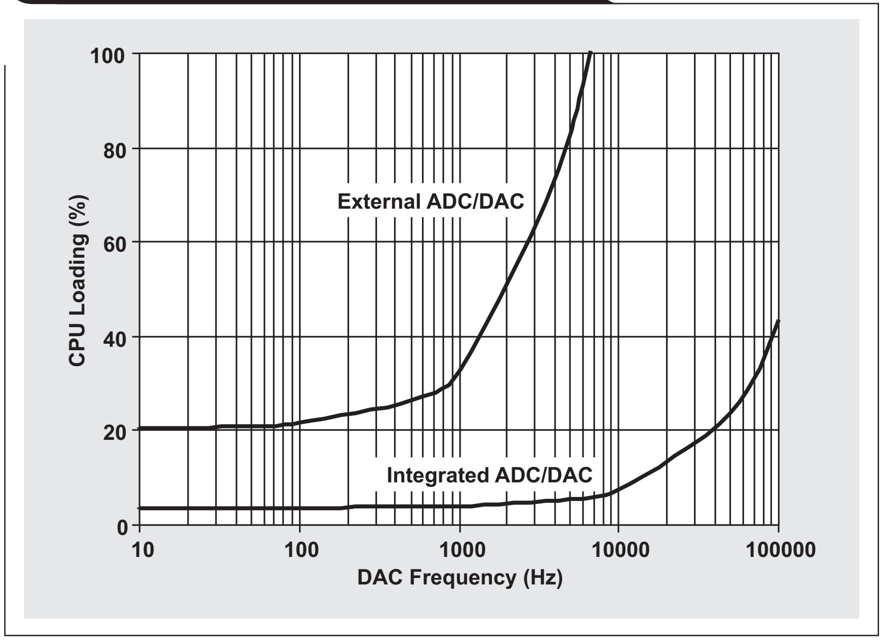
In addition to an increase in available CPU performance, the distortion effects of the sine wave output as shown for the external peripheral case are eliminated. Because the DMA handles 100% of the DAC12 updating duties, the CPU instruction execution time is not dependent on the DAC12_0 output frequency. The timing requirements for the external peripheral case relating to the DAC as a result of external ADC ISR handling are no longer valid and do not apply when determining the maximum DAC12 output frequency. If the update frequency of the DAC12 causes the CPU loading to exceed the remaining 7.65 MIPS available, only the ADC ISR handling will be affected, not the frequency of the DAC output.

As an example, channel 0 of the DAC12 module could theoretically be updated every 2 MCLKs. The maximum theoretical DAC output frequency for a 16-point sine wave is:

$$\frac{2 \text{ MCLKs}\times 16 \text{ points}}{8 \text{ MHz}} = 4\ \mu\text{s or } 250 \text{ kHz}$$

However, triggering the DMA in single-word mode requires a minimum of 4 clock cycles to complete. To transfer the DAC data, 2 clocks are needed; and 2 clocks are required to synchronize the DMA transfer to the timing of the bus. The method of using Timer_A and the DMA to update the DAC12 yields a maximum DAC output frequency

Figure 10. CPU loading performance summary



of 125 kHz. CPU loading is effectively 100%, since this leaves no MCLKs for the CPU to execute instructions and, as a result, ADC12 ISR servicing cannot be executed by the CPU. A summary of CPU loading versus DAC frequency for the discrete and integrated system scenarios is shown in Figure 10.

The two different approaches discussed here are not intended to show an ideal implementation for a given system but rather to compare and contrast the effects of chip-level integration in the mixed-signal world. For any number of reasons, some real-world systems may not be able to take full advantage of such features; but, in general, efficient utilization of MCU on-chip peripherals greatly reduces CPU loading and can also allow for more flexible control of the mixed-signal chain. As has been shown, using an external ADC and DAC can require a significant amount of dedicated CPU cycles simply to move data to and from the data ports of each device. Integration of the analog blocks of a system has a solid impact on driving the total application to a higher level of performance.

Not only do on-chip peripherals facilitate a less complex system, but often a smaller and lower-cost solution can be realized through system integration. In this example, removal of the serial communication links between the ADC and DAC also increases total achievable system performance; and CPU resources that were required for data I/O can now be used for other system control and processing tasks.

The use of integrated analog peripherals is not always feasible in a given application and is ultimately at the discretion of the engineer leading the design. In cases where an MCU such as the MSP430F169 can be used, the benefits of integrated analog become a powerful tool in enabling complex signal-chain applications.

References

For more information related to this article, you can download an Acrobat Reader file at www-s.ti.com/sc/techlit/*litnumber* and replace *"litnumber"* with the **TI Lit. #** for the materials listed below.

Document Title	TI Lit. #
1. "MSP430x1xx Family," User's Guide	.slau049
2. "MSP430x13x, MSP430x14x, MSP430x14x1 Mixed Signal Microcontroller," Data Sheet . . .	.slas272
3. "MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller," Data Sheet . . .	.slas368
4. Mark Buccini, "Using SPI synchronous communication with data converters—interfacing the MSP430F149 and TLV5616," *Analog Applications Journal* (February 2001), pp. 7–10	.slyt026
5. Mark Buccini, "Intelligent sensor system maximizes battery life: Interfacing the MSP430F123 Flash MCU, ADS7822, and TPS60311," *Analog Applications Journal* (1Q 2002), pp. 5–9	.slyt029

Related Web sites

analog.ti.com

www.ti.com/sc/device/*partnumber*

Replace *partnumber* with ADS7822, MSP430F149, MSP430F169 or TLV5616