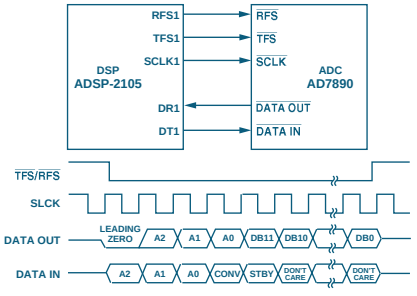
Ask The Applications Engineer—19

INTERFACING TO SERIAL CONVERTERS—I

by Eamon Nash

Q. I need data converters to fit in a tight space, and I suspect that a serial interface will help. What do I need to know to choose and use one?

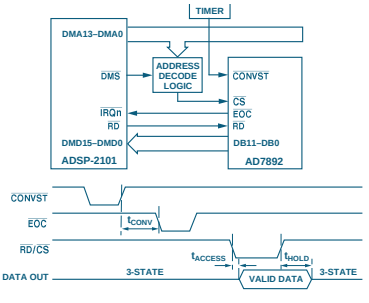
A. Let's start by looking at how a serial interface works and then compare it to a parallel interface. In doing this we will dispel some myths about serial data converters.



The figure shows an AD7890 8-Channel multiplexed 12-bit serial A/D converter (ADC) connected to the serial port of an ADSP-2105 digital signal processor (DSP). Also shown is the timing sequence that the DSP uses to communicate with the ADC. The 12 bits that constitute the conversion result are transmitted as a serial data stream over a single line. The data stream also includes three additional bits that identify the input channel that the AD7890's multiplexer is currently selecting. To distinguish the bits of the serial data stream from one another, a clock signal (SCLK) must be provided, usually by the DSP; However, sometimes the ADC supplies this clock as an output. The DSP usually (but not always) supplies an additional framing pulse that is active either for one cycle at the beginning of the communication or, as shown (TFS/RFS), for the duration of the transmission.

In this example, the DSP's serial port is used to program an internal 5-bit register in the ADC. The register's bits control such functions as selecting the channel to be converted, putting the device in power-down mode, and starting a conversion. It should be evident that the serial interface, in this case, must be bi-directional.

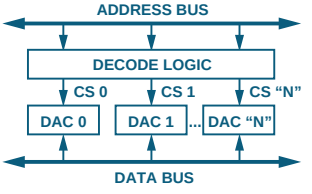
A parallel ADC, on the other hand, connects directly (or possibly through buffers) to the data bus of the processor it is interfaced with. The figure shows the AD7892 interfaced to an ADSP-2101. When a conversion is complete, the AD7892 interrupts the DSP, which responds by doing a single read of the ADC's decoded memory address.



The key difference between serial and parallel data converters lies in the number of interface lines required. From a space saving point of view, serial converters offer a clear advantage because of reduced device pin-count. This makes it possible to package a 12-bit serial ADC or DAC in an 8-pin DIP or SO package. More significantly, board space is saved because serial interface connections require fewer PCB tracks.

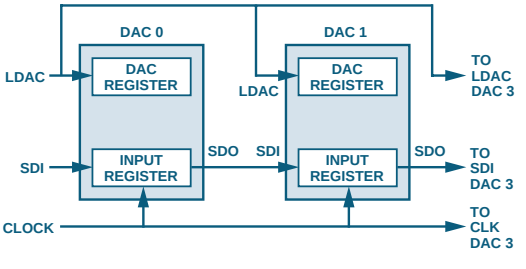
Q. My digital-to-analog converters have to be physically remote from the central processor and from one another. What is the best way to approach this?

A. Initially, you must decide whether to use serial or parallel DACs. With parallel DACs, you could map each one into a memory mapped I/O location, as shown in the figure. You would then program each DAC by simply doing a Write command to the appropriate I/O location. However, this configuration has a significant disadvantage. It requires a parallel data bus, along with some control signals, to all of the remote locations. Clearly, a serial interface, that can have as few as two wires, is much more economical.



Serial converters cannot in general be mapped into a processor's memory. But a number of serial DACs could be connected to the serial I/O port of the processor. Then, other ports on the processor could be used to generate Chip Select signals to enable the DACs individually. The Chip Select signals will require a line from each device to the interface. But there may be a limit to the number of lines on the processor that can be configured to transmit Chip Select signals.

One way of getting around this problem is to use serial DACs that can be daisy-chained together. The figure shows how to connect multiple DACs to a single I/O port. Each DAC has a Serial Data Out (SDO) pin that connects to the Serial Data In (SDI) pin of the next DAC in the chain. LDAC and SCLK are fed in parallel to all the DACs in the chain. Because the data clocked into SDI eventually appears at SDO (N clock cycles later), a single I/O port can address multiple DACs. However, the port must output a long data stream (N bits per DAC times the number of devices in the chain). The great advantage of this configuration is that device decoding is not needed. All devices are effectively at the same I/O location. The main drawback of daisy chaining is accessibility (or latency). To change the state of even a single DAC, the processor must still output a complete data stream from the I/O port.

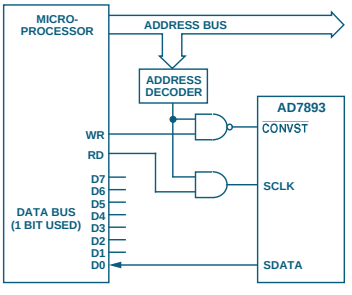


Q. If serial data converters save so much space and wire, why aren't they used in every space-sensitive application?

A. A major disadvantage of serial interfacing is the tradeoff of speed for space. For example, to program a parallel DAC, just place the data on the data bus and clock it into the DAC with a single pulse. However, when writing to a serial DAC, the bits must be clocked in sequentially (N clock pulses for an N-bit converter) and followed by a Load pulse. The processor's I/O port spends a relatively large amount of time communicating with a serial converter. Consequently, serial converters with throughput rates above 500 ksps are uncommon.

Q. My 8-bit processor doesn't have a serial port. Is there a way to interface a serial 12-bit ADC like the AD7893 to the processor's parallel bus?

A. It can of course be done using an external shift register, which is loaded serially (and asynchronously), then clocked into the processor's parallel port. However, if the sense of the question is "without external logic", the serial ADC can be interfaced as if it were a 1-bit parallel ADC. Connect the converter's SDATA pin to one of the processor's data bus lines (it is connected to D0 in the diagram). Using some decode logic, the converter can be mapped into one of the processor's memory locations so that the result of the conversion can be read with 12 successive Read commands. Then additional software commands integrate the LSBs of the 12 bytes that were read into a single 12-bit parallel word.



This technique, which is sometimes referred to as "bit banging", is very inefficient from a software perspective. But it may be acceptable in applications in which the processor runs much faster than the converter.

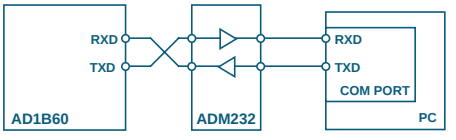
Q. In the last example, a gated version of the processor's write signal was used to start conversions on the AD7893. Are there problems with that approach?

A. I am glad you spotted that. In this example, a conversion can be initiated by doing a dummy write to the AD7893's mapped memory location. No data is exchanged, but the processor provides the write pulse needed to begin the conversion. From a hardware perspective, this configuration is very simple because it avoids the need to generate a conversion signal.

However, the technique is not recommended in ac data-acquisition applications, in which signals must be sampled periodically. Even if the processor is programmed to do periodic writes to the ADC, phase jitter on the Write pulse will seriously degrade the attainable signal-to-noise ratio (SNR). The gating process may make the Write signal jitter even worse. A sampling clock phase jitter level of as little as 1 ns, for example, would degrade the SNR of an ideal 100-kHz sine wave to about 60 dB (less than 10 effective bits of resolution). There is also an additional danger that overshoot and noise on the sampling signal will further degrade the integrity of the analog to digital conversion.

Q. When should I choose a converter with an asynchronous serial interface?

A. An asynchronous link allows devices to exchange unclocked data with each other. The devices must initially be programmed to use the identical data formats. This involves setting a particular data rate (usually expressed in baud, or bits per second). A convention, that defines how to initiate and end transmissions, is also necessary. We do this using identifiable data sequences called *start* and *stop* bits. The transmission may also include *parity* bits that facilitate error detection.



The figure shows how the AD1B60 Digitizing Signal Conditioner interfaces to a PC's asynchronous COM Port. This is a 3-wire bidirectional interface (the ground lines have been omitted for clarity). Notice that the receive and transmit lines exchange roles at the other end of the line.

An asynchronous data link is useful in applications in which devices communicate only sporadically. Since start and stop bits are included in every transmission, a device can initiate communication at any time by simply outputting its data. The number of connections between devices is reduced because clocking and control signals are no longer necessary.

Q. The data sheet of an ADC I am considering recommends using a non-continuous clock on the serial interface. Why?

A. The specification probably requires that the clock be kept inactive while the conversion is in progress. Some ADCs require this because a continuous data clock can feed through to the analog section of the device and adversely affect the integrity of the conversion. A continuous clock signal can be discontinued during conversion if the I/O port has a framing pulse; it is used as a gating signal that enables the serial clock to the converter only during data transfer.

Q. What makes a device SPI or MICROWIRE compatible?

A. SPI (Serial Peripheral Interface) and MICROWIRE are serial interface standards developed by Motorola and National Semiconductor, respectively. Most synchronous serial converters can be easily interfaced to these ports; but in some cases additional "glue" logic may be necessary.

Q. O.K. I decided to put prejudice aside and use a serial ADC in my current design. I have just wired it up as the data sheet specifies. When my micro reads the conversion result, the ADC always seems to output FFF_{HEX}. What's happening?

A. Perhaps you are having a communications problem. We need to look at the connections between the ADC and the processor—and at how the timing and control signals have been set up. We also need to look at the Interrupt structure. The next installment will return to this issue, discussing the problems encountered when designing serial interfaces.

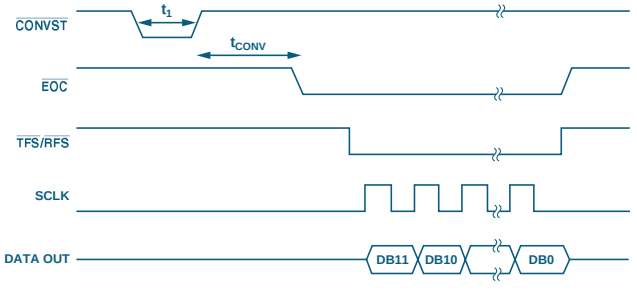
Ask The Applications Engineer—20

INTERFACING TO SERIAL CONVERTERS—II

by Eamon Nash

Q. At the end of our discussion in the last issue, I was having a problem establishing communication between my ADC and my microcontroller. If you recall, the microcontroller always seemed to be reading a conversion result of FFF_{HEX} regardless of the voltage on the analog input. What could be causing this?

A. There are a number of possible timing-related error sources. You could start trouble-shooting this problem by connecting all of the timing signals either to a logic analyzer or to a multi-channel oscilloscope (at least three channels are needed to look at all signals simultaneously). What you would see on the screen would look similar to the timing diagram in the figure below. First make sure that a Start Conversion command (CONVST) is being generated (coming either from the micro or from an independent oscillator). A frequent mistake is to apply a CONVST signal with the wrong polarity. The conversion is still performed, but not when you expect it to be. It is also important to remember that there is usually a minimum pulse width requirement on the CONVST signal (typically about 50 ns). The standard Write or Read pulse from fast microprocessors may not satisfy this requirement. If too short, the pulse width can be extended by inserting software Wait states.



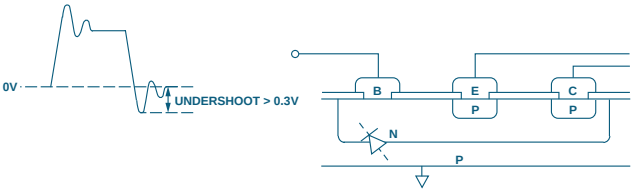
Make certain that the micro is waiting for the conversion to be completed before the Read cycle begins. Your software should either be taking note of the time required to convert or be waiting for an End of Conversion (EOC) indicator from the ADC to generate an interrupt in the micro. Make sure that the polarity of the EOC signal is correct, otherwise the ADC will cause an interrupt while the conversion is in progress. If the micro is not responding to the interrupt, you should examine the configuration of the interrupt in your software.

It is also important to consider the state of the serial clock line (SCLK) while it is not addressing the converter. As I mentioned in our previous discussion, some DACs and ADCs do not operate correctly with continuous serial clocks. In addition to this, some devices require that the SCLK signal always idles in one particular state.

Q. O.K. I've found and corrected some bugs in my software and things seem to be improving. The data from the converter are changing as I vary the input voltage but the conversion results seem to have no recognizable format.

A. Once again there are a number of possible error sources. The ADC will be outputting its conversion result either in straight binary or in twos complement format (BCD data converters are no longer widely used). Check that your micro is configured to accept the appropriate format. If the micro can't be configured to accept twos complement directly, you can convert the data to straight binary by exclusive-or'ing the number with 100 . . . 00 binary.

Normally the leading edge of the serial clock (either rising or falling) will enable the data out of the ADC and onto the data bus. The trailing edge then clocks the data into the micro. Make sure that both micro and ADC are operating under the same convention and that all Setup and Hold times are being met. A conversion result that is exactly half or double what one would expect is a tell-tale sign that the data (especially the MSB) is being clocked on the wrong edge. The same problem would manifest itself in a serial DAC as an output voltage that is half or double the expected value.

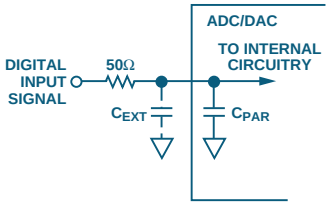


The digital signals driving the converter should be clean. In addition to causing possible long-term damage to the device, overshoot or undershoot can cause conversion and communication errors. The figure shows a signal with a large overshoot spike driving the clock input of a single-supply converter. In this case, the clock input drives the base of an PNP transistor. As is usual practice, the P-type substrate of the device is internally connected to the most negative potential available—in this case, ground. An excursion of more than 0.3 volts below ground on the SCLK line is sufficient to begin turning on a parasitic diode between the N-type base and the P-type substrate. If this happens frequently, over the long term, it may lead to damage to the device.

In the short term, though not causing damage, energizing the normally inert substrate affects other transistors in the device and can lead to multiple clock pulses being detected for each pulse applied. The resulting jitter is a serious matter in serial converters—but is less of a problem in parallel converters, because the Read and Write cycles generally depend upon the first applied pulse; subsequent pulses are ignored. However, the noise performance on both serial and parallel converters can suffer if signals of this kind are present during conversion.

The figure shows how overshoot can be easily reduced. A small resistor is placed in series on the digital line that is causing the problem. This resistance will combine with C_{par}, the parasitic capacitance of the digital input, to form a low-pass filter which should eliminate any ringing on the received signal. Typically a 50-Ω resistor is recommended, but some experimentation may be necessary. It may also be necessary to add an external capacitance from the input to ground if the internal capacitance

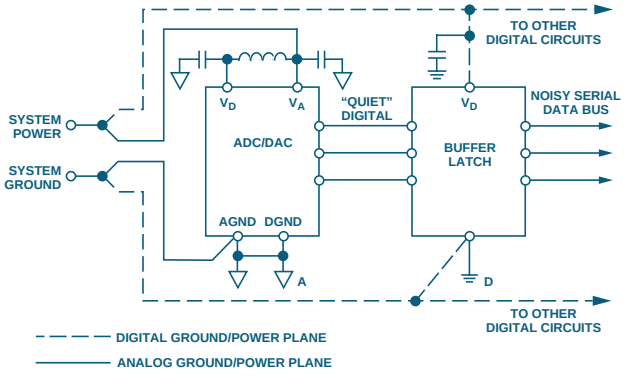
of the digital input is insufficient. Here again, experimentation is necessary—but a good starting point would be about 10 pF.



Q. You mentioned that clock overshoot can degrade the noise performance of a converter. Is there anything else I can do from an interfacing point of view to get a good signal to noise ratio?

A. Because your system is operating in a mixed-signal environment (i.e., analog and digital), the grounding scheme is critical. You probably know that—because digital circuitry is noisy—analog and digital grounds should be kept separate, joined at only one point. This connection is usually made at the power supply. In fact, if the analog and digital devices are powered from a common supply, as might be the case in a +5 V or +3.3 V single-supply system, there is no choice but to connect the grounds back at the supply. But the data sheet for the converter probably has an instruction to connect the pins AGND and DGND at the device! So how can one avoid creating a ground loop that can result if the grounds are connected in two places?

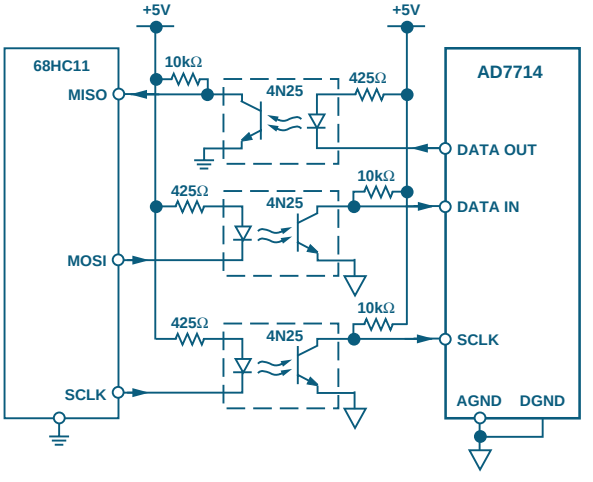
The figure below shows how to resolve this apparent dilemma. The key is that the AGND and DGND labels on the converter's pins refer to the parts of the converter to which those pins are connected. The device as a whole should be treated as *analog*. So after the AGND and DGND pins have been connected together, there should be a single connection to the system's analog ground. True, this will cause the converter's digital currents to flow in the analog ground plane, but this is generally a lesser evil than exposing the converter's DGND pin to a noisy digital ground plane. This example also shows a digital buffer, referred to digital ground, to isolate the converter's serial data pins from a noisy serial bus. If the converter is making a point-to-point connection to a micro, this buffer may be unnecessary.



The figure also shows how to deal with the increasingly common challenge of powering a mixed-signal system with a single power supply. As in the grounding case, we run separate power lines (preferably power planes) to the analog and digital portions of the circuit. We treat the digital power pin of the converter as analog. But some isolation from the analog power pin, in the form of an inductor, is appropriate. Remember that both power pins of the converter should have separate decoupling capacitors. The data sheet will recommend appropriate capacitors, but a good rule of thumb is 0.1 µF. If space permits, a single 10-µF capacitor per device should also be included.

Q. I want to design an isolated serial interface between an ADC and a microcontroller using opto-isolators. What should I be aware of when using these devices?

A. Opto-isolators (also known as opto-couplers) can be used to create a simple and inexpensive high-voltage isolation barrier. The presence of a galvanic isolation barrier between converter and micro also means that analog and digital system grounds no longer need to be connected. As shown in the figure, an isolated serial interface between the AD7714 precision ADC and the popular 68HC11 microcontroller can be implemented with as few as three optoisolators.



The designer should be aware, though, that the use of optoisolators having relatively slow rise and fall times with CMOS converters can cause problems, even when the serial communication is running at a slow speed.

CMOS logic inputs are designed to be driven by a definite logic zero or logic one. In these states, they source and sink a minimal amount of current. However, when the input voltage is in transition between logic zero and logic one (0.8 V to 2.0 V), the gate will consume an increased amount of current. If the opto-isolators used have relatively slow rise and fall times, the excessive amount of time spent in the dead-band will cause self-heating in the gate. This self-heating tends to shift the threshold voltage of the logic gate upwards, which can lead to a single clock edge being interpreted by the converter as multiple clock pulses. To prevent this threshold jitter, the lines coming from the optoisolators should be buffered using Schmitt trigger circuits, to deliver fast, sharp edges to the converter.