WP279 (v1.0) July 18, 2008

Digitally Removing a DC Offset: DSP Without Mathematics

By: Ken Chapman

Many designers feel uncomfortable at the mention of digital signal processing (DSP). The approach to DSP adopted in this white paper is intended to instill confidence in designers who are implementing DSP algorithms in their designs, resulting in more efficient hardware implementations. Rather than beginning with mathematical descriptions of a system, designers can learn how simple functions in the analog world can be modeled and then can convert those functions into digital representations.

This white paper examines how to remove the DC content from a digitally sampled waveform using DSP without complicated mathematics. The first half of the white paper examines DSP in a qualitative manner and illustrates how to create a circuit capable of performing the required signal processing. The second half describes how to optimize the derived function for use in audio telecommunications applications using SRL16Es.

Creating the Circuit

This section of the white paper presents a brief discussion of digital signal processing and describes a method of creating a circuit that can perform DSP. The circuit is implemented using a Spartan®-3 generation device, although Virtex® devices can be used as well.

Sampled Waveforms

DSP utilizes digital samples, which are numbers that represent the amplitude of a waveform and are taken at regular intervals. These samples are normally the result of an Analog-to-Digital Converter (ADC) that generates values of a given number of bits (resolution) at a sample rate set by a sample clock.

In the upper plot of Figure 1, an analog waveform is applied to the ADC. The input signal should remain within the specified input voltage swing of the ADC (in this case, ±1 volt). The ADC samples this waveform at a frequency (f_S) that is relatively fast in comparison to the frequency content of the signal. (Nyquist theory requires the waveform to be sampled at a rate at least twice that of the highest frequency present.)

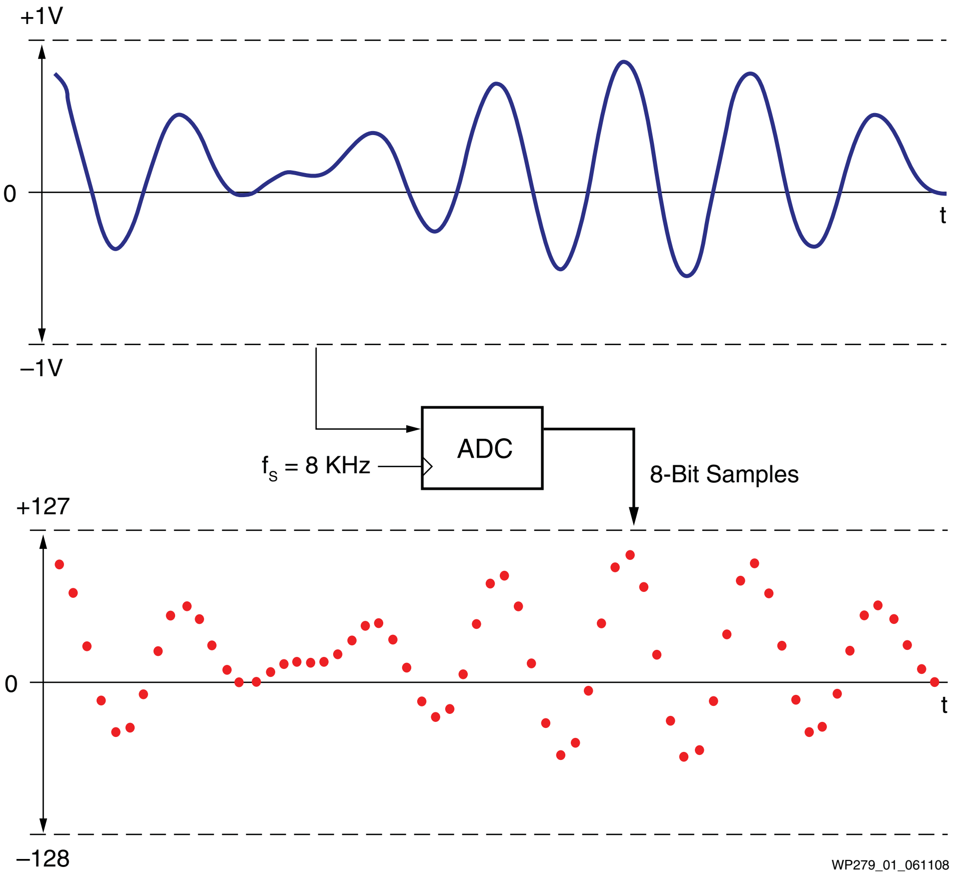

Figure 1: **Test Signal Formed of 800 Hz and 960 Hz Components with a DC Offset**

In the lower plot of Figure 1, the actual digital samples provided by the ADC are shown as red dots. In this plot, the most important part of DSP has occurred. In this case, the samples are represented by 8-bit numbers. A two's complement format is used to represent both positive and negative quantities. The first values of the plot are +104, +80, +31, –19, –48, and –44. These can be represented in hexadecimal as `68`, `50`, `1F`, `ED`, `D0`, and `D4`. Thus, DSP essentially involves working with a stream of numerical data and manipulating it in some way.

Although the only information available to work with are the values represented by the red dots, the shape of the waveform can be made more apparent by joining the dots with lines. Although an analog waveform then becomes more apparent, it would be less pure than it originally was (this forms the essence of quantization noise).

The waveform plots in Figure 2 contain high frequency components of some kind (800 Hz and 960 Hz in this test case). The waveform spends a greater percentage of time above the zero axis than it does below it, indicating some kind of positive DC bias. This is clearly seen in the digital plot because there are many more positive red dots than negative.

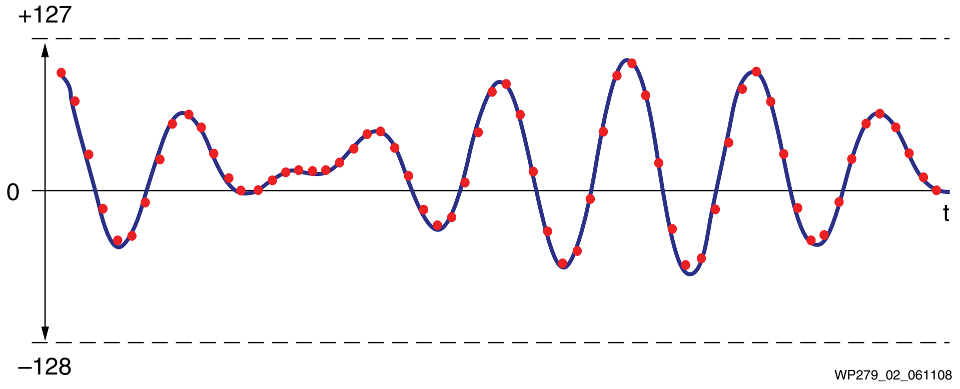

Figure 2: **Digital Samples Joined Together to Form an Analog Signal**

Generally speaking, a DC offset such as this is undesirable because it means that the positive peaks of the waveform are more likely to exceed the maximum level that can be represented. In the ideal world, the DC offset would be removed before the analog-to-digital (A/D) conversion; however, this can be difficult to achieve. Indeed, the analog components could have unintentionally inserted the DC bias as part of the signal amplification and conditioning.

Removing the DC Offset

Given that a DC offset has a frequency of zero, the DC offset can, theoretically, be removed by the use of a high-pass filter. This can lead to full-scale DSP and investigation of such things as Finite Impulse Response (FIR) filters. However, this white paper takes a more empirical approach to solving the problem of DC offsets by avoiding as much math as possible and finding a much easier and cost-effective implementation.

If the DC offset level is known, it is possible to remove it with a simple subtraction. In the example discussed here, the DC offset of the digital samples is determined to be +19. Thus, each input sample from the ADC must have a value of +19 subtracted from it. The output from the subtracter is the waveform without any DC offset (Figure 3). This is an example of DSP, because a stream of digital samples has been manipulated to form new samples. The value of the first sample is +104 – 19 = +85.

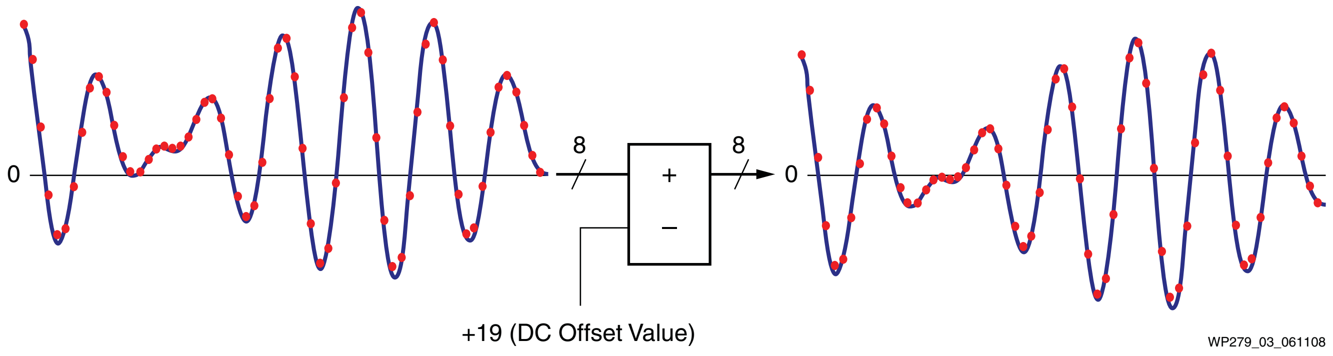

Figure 3: **Digital Samples Input into a Subtracter to Remove DC Offset**

Thus, it is seen that a subtracter can be an important DSP function. (The subtracter is also very well-supported by Virtex and Spartan devices.) As with the basic adder function, each slice of the configurable logic block (CLB) can implement a 2-bit subtracter. Therefore, a simple 8-bit subtracter requires four slices. Because this is such a basic function, it is supported well in many design flows, including HDL and System Generator.

Finding the DC Level

Although the DC offset was removed with the subtracter, the process relied on an arbitrary DC value. Therefore, a method of automatically deriving the DC offset value needs to be determined. Although this is a little more complicated, an empirical approach can again be taken to finding a solution with another very common function, and thereby avoid a lot of DSP theory.

In the analog world, the simplest way to find the average DC level of a signal is to smooth it with a capacitor (Figure 4). The larger the value of the smoothing capacitor, the steadier the DC level, especially if there is a load current.

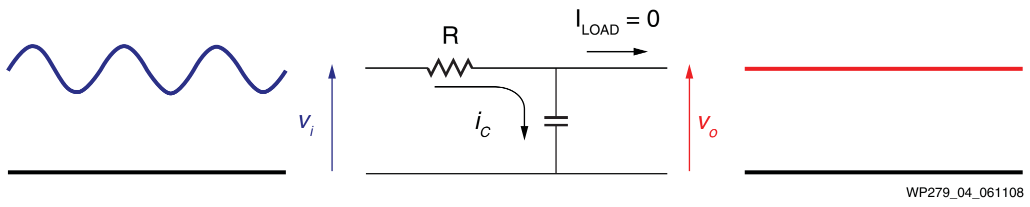

Figure 4: **RC Circuit Used to Determine the DC Level of a Digital Signal**

In theory, a differential equation must be used to solve this simple circuit. However, if only an instant in time is considered, then simple linear equations can be used.

The voltage across the resistor R is given by $v_i - v_o$. Equation 1 defines the current i_c flowing into the capacitor.

$$i_c = \frac{v_i - v_o}{R} \qquad \textit{Equation 1}$$

If the input voltage is higher than the average value v_o, the capacitor charges. Likewise, if the input voltage is lower than the average value, the current is negative (flowing out of the capacitor), and the capacitor discharges.

Equation 2 provides the linear equation for the charge Q on a capacitor.

$$Q = C \times V = I \times T \qquad \textit{Equation 2}$$

So, for a constant current I for a period of time T, the voltage V on the capacitor rises by an amount indicated by Equation 3.

$$V = \frac{I \times T}{C} \qquad \text{Equation 3}$$

The larger the value of C, the smaller the change in voltage for a given current and time. (∇ means "change in.") This means that a final formula can be derived that describes this simple RC circuit (Equation 4).

$$\nabla V_o = \frac{\nabla T}{R \times C} \times (v_i - v_o) \qquad \text{Equation 4}$$

Equation 4 is actually a simple linear equation. During a period of time ∇T, the voltage across the capacitor changes by an amount proportional to the difference between the input and output voltages. This equation is only valid if the duration of time ∇T is so small that the voltage change ∇V_o does not significantly change the value of $v_i - v_o$.

By using a constant value k, which is set by the combination of R, C, and the period of time ∇T over which each calculation is made, Equation 4 can be simplified to Equation 5.

$$\nabla V_o = k \times (v_i - v_o) \qquad \text{Equation 5}$$

At the end of each period, the output voltage becomes the previous value of v_o plus the incremental value ∇V_o (Equation 6).

$$V_o' = V_o + k \times (v_i - v_o) \qquad \text{Equation 6}$$

Where $k = \nabla T/(R \times C)$.

A simple experiment can be tried to prove this formula. If $R = 50\Omega$, $C = 100\ \mu F$ (a time constant of $R \times C = 5$ ms), and $\nabla T = 1$ ms, then $k = 0.2$. By applying a simple 10V step input, the calculated output for each step of time generates the exponential charge curve expected for such an RC circuit, as shown in Figure 5.

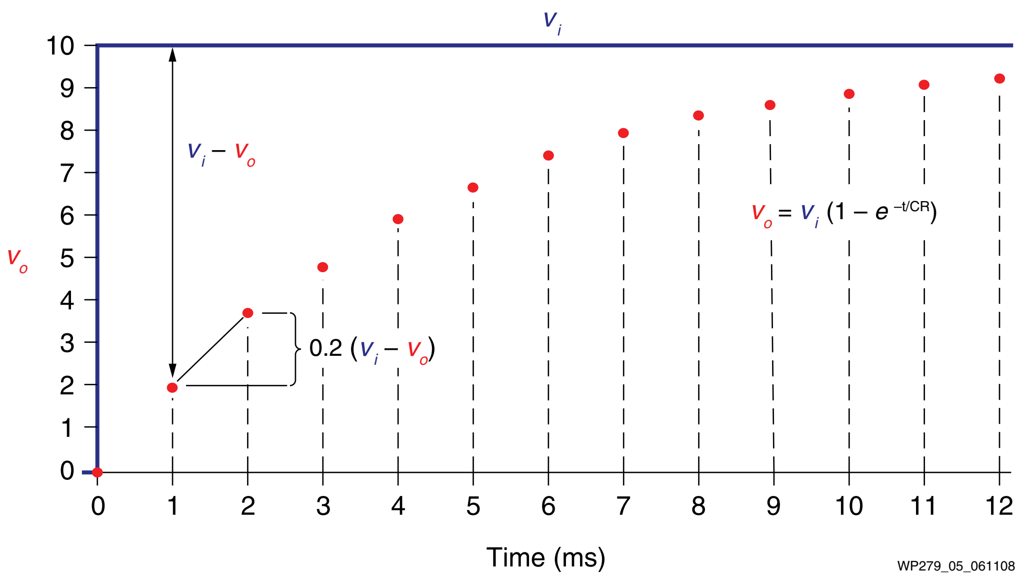

Figure 5: **Exponential Charge Curve for RC Circuit**

Of interest is that a set of values has been calculated at regular intervals using a simple linear equation to describe a complex differential function. Because the points are calculated at regular intervals, digital signal processing has been performed to emulate an analog function.

Representing a smoothing capacitor RC circuit, Equation 6 is now easy to realize as a digital circuit. The clock sets the sample rate to a register that holds the current charge voltage value, with the clock frequency determining the period ∇T for each step. The rest of the equation is then formed by a subtracter, an adder, and a multiplier, as shown in Figure 6.

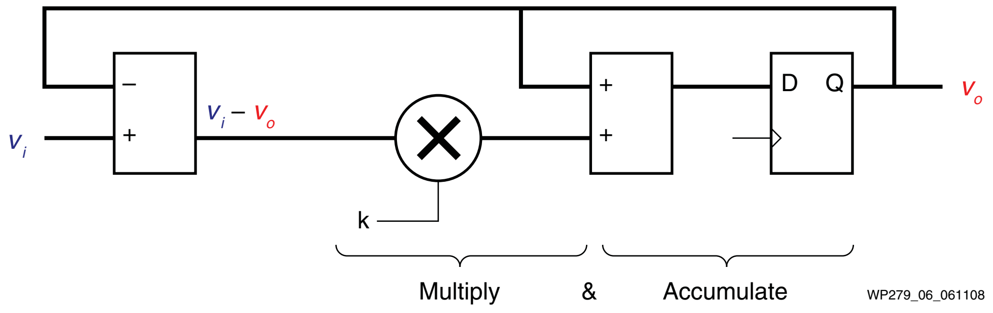

Figure 6: **Digital RC Circuit**

The major part of this circuit consists of a multiplier feeding an accumulator. This forms the most common building block in DSP, which is often called a multiply-and-accumulate (MAC) block. The value of the coefficient k sets the behavior of the circuit. Given that $k = \nabla T/(R \times C)$, and that ∇T has been fixed by setting the clock rate, the value of k is inversely proportional to the RC time constant being modeled.

The circuit is now connected to the original waveforms that contain DC offsets. The sample rate is 8 KHz (∇T = 125 µs). The resulting exponential charge curves are shown for $k = 1/32$ (Figure 7) and $k = 1/256$ (Figure 8). Thus, these figures show that using a smaller k value (the equivalent of a larger RC time constant) results in a longer charging period but a smoother output. As expected, there is a trade-off between the time taken to reach the DC offset level and the smoothness of the final value derived.

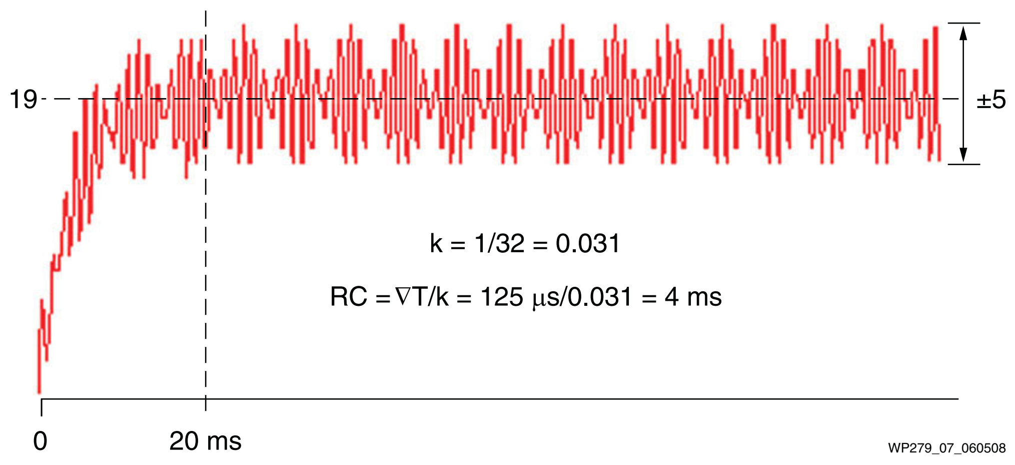

Figure 7: **Exponential Charge Curve for k = 0.031**

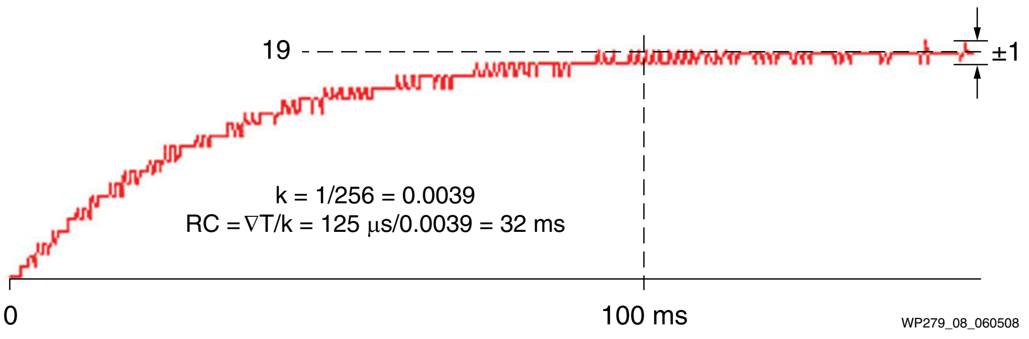

Figure 8: **Exponential Charge Curve for k = 0.0039**

Implementing the Fractional Coefficient Multiplier

Although the addition and subtraction blocks are simple to implement, the multiplier block can be more challenging. Synthesis tools can automatically create multipliers from logic or target the dedicated multipliers within the devices; however, the function still needs to be described appropriately and creation of unnecessary logic must be prevented. The requirement in this circuit is to take an integer value ($v_i - v_o$) and multiply it by a value *k* that is less than one, as is common in DSP designs.

Fixed-point arithmetic is the solution; however, it involves interpreting the bits of integer arithmetic in a different manner. Consider the effect of multiplying an 8-bit integer by the binary pattern `0101`.

```
   7 7              0 1 0 0 1 1 0 1
 ×   5                  ×   0 1 0 1
 -----      -------------------------
 3 8 5      0 0 0 1 1 0 0 0 0 0 0 1
```

In this case, the `0101` pattern is the integer value +5_{10}, and the multiplication is straightforward. The multiplication of an 8-bit number and a 4-bit number forms a potential 12-bit product.

```
   7 7              0 1 0 0 1 1 0 1
 ×   1 . 2 5        ×         0 1 . 0 1
 -----------      -----------------------
   9 6 . 2 5      0 0 0 1 1 0 0 0 0 0 . 0 1
```

Now the same binary pattern (`0101`) has been divided by 4 because a binary point was inserted two bits from the right-hand side of the number, forming the value 1.25_{10}. However, the binary pattern of the product is exactly the same as in the previous integer case.

Hence, the multiplier works exactly the same way with fractional numbers as it does with integers. To interpret the product result correctly, a binary point is inserted the same number of bits from the right-hand side as in the input value. The bits to the left of the point (`1100000`) represent the integer part of the result (96_{10}), and the bits to the right of the point (`01`) represent the fractional part of the result (0.25_{10}).

```
   7 7                    0 1 0 0 1 1 0 1
 ×   0 . 0 7 8 1 2 5      ×         0 . 0 0 0 1 0 1
 -------------------      -----------------------
     6 . 0 1 5 6 2 5          0 0 0 1 1 0 . 0 0 0 0 0 1
```

In this last case, the binary point is located 6 bits away from the right-hand side of the original `0101` pattern. This divides the value by $2^6 = 64$. It appears that the value must increase by 3 bits just to position the binary point. However, the binary pattern of the product is exactly the same as it was before and simply needs to be interpreted correctly. The binary point is located 6 bits from the right-hand side; hence, the integer part to the left of the point is 6_{10}, and the fractional part to the right of the point is 0.015625_{10} (1/64).

The options available for implementing the multiplier in Virtex and Spartan devices are:

- All current Virtex and Spartan devices have dedicated multipliers, with some having full DSP blocks. These features should be used unless there are more pressing uses for them in the rest of the system design. Given the full variable nature of these multipliers, the value of *k* can be changed during the acquisition process. By starting with a large value for *k*, the circuit rapidly locates the approximate DC level. If smaller values of *k* are used, a very stable and smooth DC value is achieved. This is like having a variable resistor in the analog RC circuit.
- A synthesis tool can be used to create a multiplier. With this option, it is worth looking at what resources have been used, particularly if the products are cost-sensitive.
- Efficient variable multipliers can be generated in the CORE Generator™ software, but a constant coefficient multiplier can also be specified that is formed of fewer slices. In this case, the value *k* is generally a constant.

The Digital RC Circuit

The fractional multiplier techniques discussed earlier can be used to create a bit-accurate model of an RC circuit and, hence, digitally detect the DC level of the input signal (Figure 9).

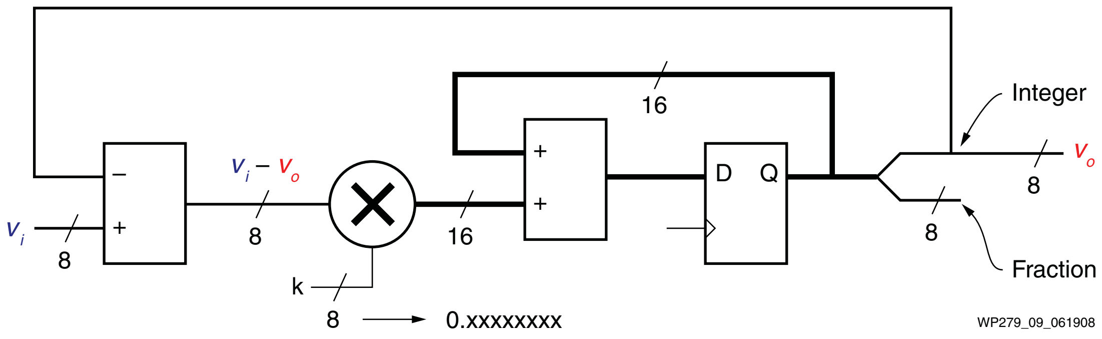

Figure 9: **Digital RC Circuit**

The input samples are 8 bits; therefore, the DC content must also lie within the same range (the DC offset should not be near the system limits). Because the DC detection circuit starts at zero, and the DC offset is likely to be a very slow-moving level, an 8-bit subtracter can be used to form the difference signal ($v_i - v_o$).

For the multiplier stage, an 8-bit × 8-bit multiplier is selected, which produces a 16-bit product. The *k* value is less than unity because all bits have been specified to the right of the binary point. The difference signal is signed (± values), but the coefficient *k* is always positive.

Hence, the range of available k values is from 0.0039_{10} (1/256) to 0.9961_{10} (255/256). Because values of 0.03 and higher would have a large ripple (as seen in Figure 7, page 6 and Figure 8, page 7), it is better to use fewer bits for the value of k, but retain the same binary bit position. However, the larger values of k can be used to rapidly reach the DC level and can then be reduced.

The 16-bit product must be considered as consisting of an 8-bit integer (left of the binary point) and an 8-bit fraction (right of the binary point). Therefore, the accumulator must work with a full 16 bits. The accumulation of the fractional values and the integer parts of the products is required to ensure that the DC level is able to adjust even when very small k values are used and very small products are being generated. Furthermore, only the integer portion of the accumulated value represented by the eight most significant bits is used to identify the DC level that is to be used by the signal-correcting subtracter described in "Removing the DC Offset," page 3.

Optimizing the Circuit

The second half of this white paper discusses methods of optimizing the high-performance circuit created in the first section. The size of the parallel circuit is first reduced. Then the SRL16E is used to efficiently create a serial version of the circuit that is suitable for low sample-rate applications. This section considers an audio communications rate of 8 KHz.

Removing the Multiplier Logic

Potentially, the largest part of the circuit so far is the multiplier. Although dedicated multipliers are available in Xilinx® devices, minimizing the use of these key resources is highly desirable for lower power consumption and cost-sensitive designs, especially when using Spartan-3 generation devices.

The multiplier can easily be removed from the example circuit using the coefficient values of $k = 1/32$ and $k = 1/256$ from the two response plots in Figure 7, page 6 and Figure 8, page 7. In both cases, the coefficient values are represented by numbers in which only one bit is active. The second coefficient value is adopted because low ripple is much more desirable than the response time, especially because even 100 ms is relatively short.

```
      7 7             0 1 0 0 1 1 0 1
  ×   0 . 0 0 3 9       ×       0 . 0 0 0 0 0 0 0 1
  ---------------   ---------------------------------
      0 . 3 0 0 3                   0 . 0 1 0 0 1 1 0 1
```

Because the multiplication process only requires that the variable input be multiplied by 1, the output product is the same as the input. Thus, there is no need for a real multiplier; the output product is the same value, and the bit width is the same as the variable input. All that is required is to apply the variable input value with the binary point reassigned to the correct position. The complete DC offset removal circuit is then reduced to the circuit shown in Figure 10.

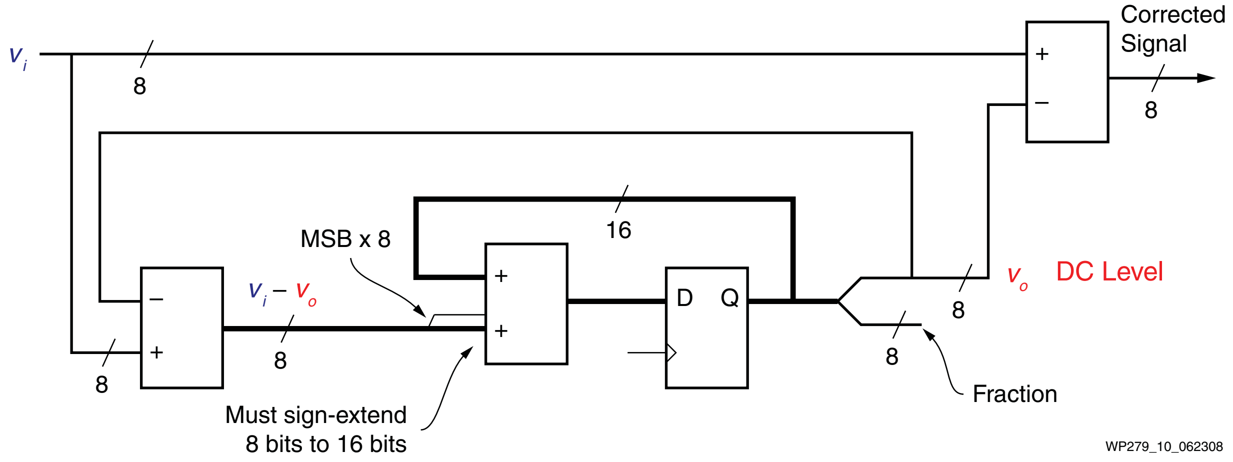

Figure 10: **Digital RC Circuit without Multiplier**

The circuit now consists only of an accumulator and two subtracters. Care is required when connecting the variable difference signal ($v_i - v_o$) to the accumulator input. The eight bits are applied to the least significant byte of the 16-bit input to represent that the eight bits are all fractional (to the right of the binary point). However, the upper byte must also be defined. This must be achieved using sign extension by replicating the MSB of the 8-bit value another eight times to form either hexadecimal `00` or `FF`. This is done so that the two's complement logic of the accumulator correctly adds both positive and negative values. With this very small *k* value, it is apparent why the accumulation of the fractional (and integer) parts must be performed.

Removing a Subtracter

Seeing the DC level detector and the DC-removing subtracter together for the first time (Figure 10), it is apparent that one of the subtracters is redundant. The corrected signal is the original signal with the DC level subtracted from it. This means that the output is the value $v_i - v_o$, which is the same as the difference signal being created by the subtracter within the DC detection circuit. This further means that the complete DC offset removal circuit can be reduced to just one accumulator and one subtracter (Figure 11).

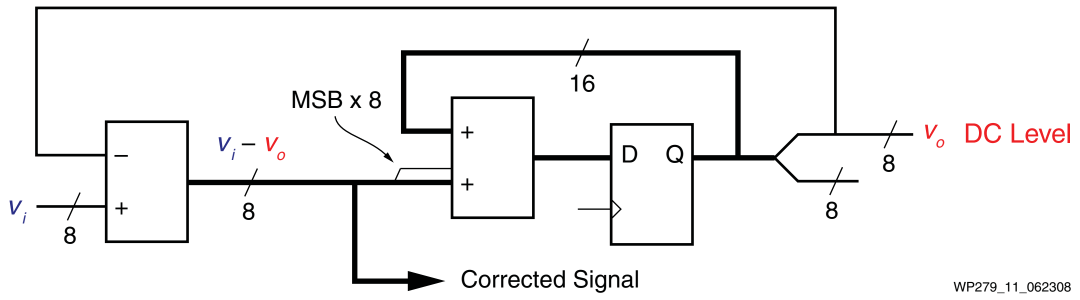

Figure 11: **Digital RC Circuit Comprising One Accumulator and One Subtracter**

Using the simple but accurate rule that a 2-bit add or subtract function fits into a slice in a Spartan-3 generation device, this circuit now requires only 12 slices. For each additional bit of sample width, the subtracter and accumulator each increase by one bit and, accordingly, increase the total size by one slice. Therefore, with 16-bit input samples, the size increases to 20 slices. As a parallel circuit, this can also support a sample rate well in excess of 100 MHz.

Low Sample-Rate Applications

Although 12 to 20 slices can seem like a small quantity, they still constitute 2–3% of the smallest XC3S50A device, and this DC offset removal can easily be seen as just a preliminary process to the main function. In a typical application, this DC offset removal can be required in a telephone conferencing facility. Each of the input lines (represented by digital samples) is ultimately summed together within the system, and the contribution of multiple small DC offsets can have an adverse effect on the overall dynamics. The requirement for a DC offset removal circuit on each line input could cause the 2–3% contribution of all devices to add up to something significant.

Also typical of audio telecommunications, data samples are transmitted serially between units as packets within data frames or even directly from the ADC. The Texas Instruments TLC320AC01C analog interface circuit (AIC) device [Ref 1] uses a serial communications protocol with 14-bit A/D samples being transmitted with the most significant bit first as part of each 16-bit transfer (Figure 12).

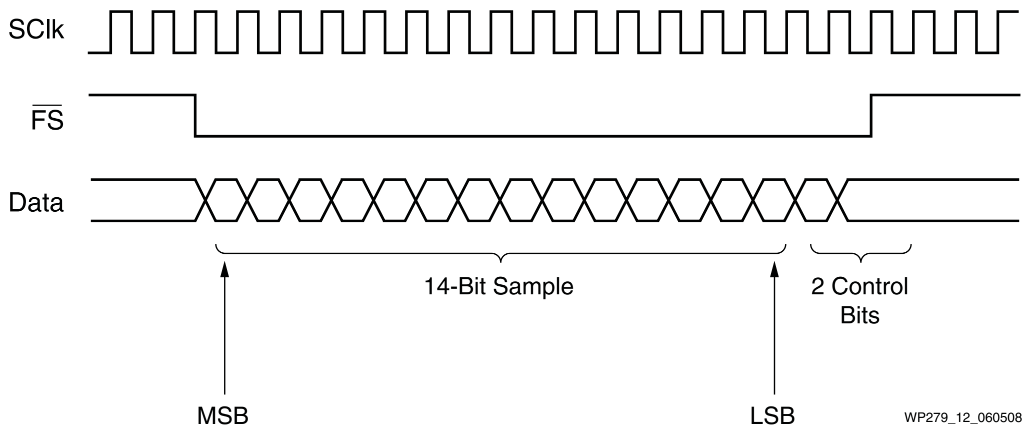

Figure 12: **TLC320AC01C Serial Communications Protocol**

To use the parallel implementation of the DC offset removal circuit, such serial data samples need to be applied to a 14-bit shift register to read the sample in parallel. This requires an additional seven slices.

Staying Serial

Instead of converting to parallel, it is better to process the data serially. Although this can take many clock cycles to achieve, with sample rates as low as 8 KHz, even a 10 MHz clock provides 1,250 clock cycles in which to implement the task.

The functions of an accumulator and a subtracter are to be achieved in the example circuit. In the serial processing form, these only have to resolve one bit of the result in each clock cycle and take on the form of a 1-bit full adder or 1-bit full subtracter. The functionality can be derived from a truth table (see Table 1). The only special observation to be made is that the process starts with the least significant bit first, and then generates the result bit and a CARRY/BORROW flag for use in the calculation of the next most significant bit of the process. During the processing of the first bit (LSB), any previous carry/borrow status must be ignored. This can be achieved by a masking signal.

Table 1: **Truth Table for Serial Processing**

MASK	C_{in}	A	B	SUM	CARRY	SUB	BORROW
0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	1
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	0	1	0	1
0	1	1	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1
1	0	1	0	1	0	1	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	0
1	1	1	1	0	1	0	0

Notes:

1. The BORROW flag has been specified as active-High and is seen as a High C_{in} during the next bit processing of A-B-C_{in}.
2. The shaded cells indicate that C_{in} is masked.

An advantage of these serial arithmetic functions is that however complex the truth table might be, the functions fit perfectly into the 4-input look-up tables of Spartan-3 generation devices. Hence, an adder or subtracter requires just one slice each. The serial adder and serial subtracter are shown in Figure 13 and Figure 14, respectively.

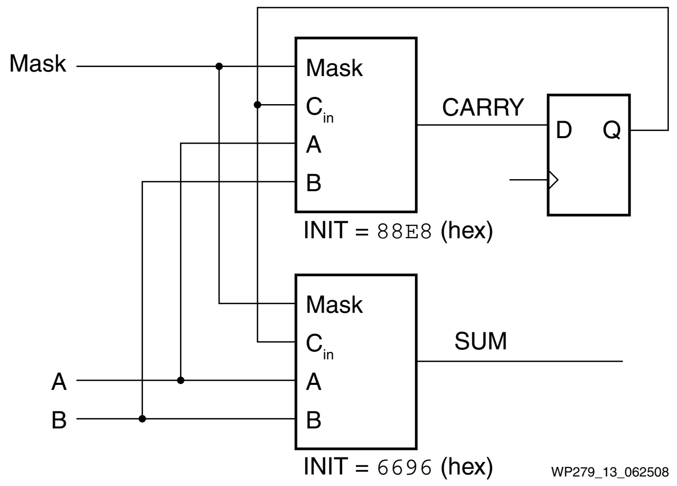

Figure 13: **Serial Adder**

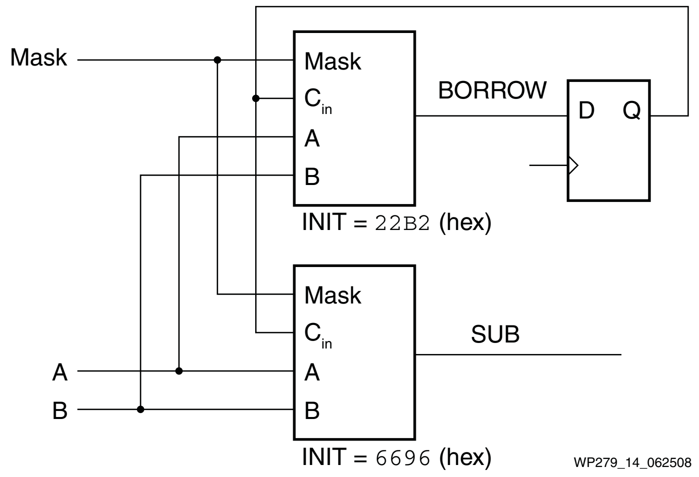

Figure 14: **Serial Subtracter**

To convert the adder into an accumulator, storage must be added for the accumulated value. In this case, the storage can be formed serially; therefore, the SRL16E becomes the obvious selection. If a 16-bit accumulator is not adequate, two SRL16E components can be used to form a complete 32-bit accumulator in only two slices (Figure 15). The clock enable can be used to freeze the contents between bursts of sample processing activity.

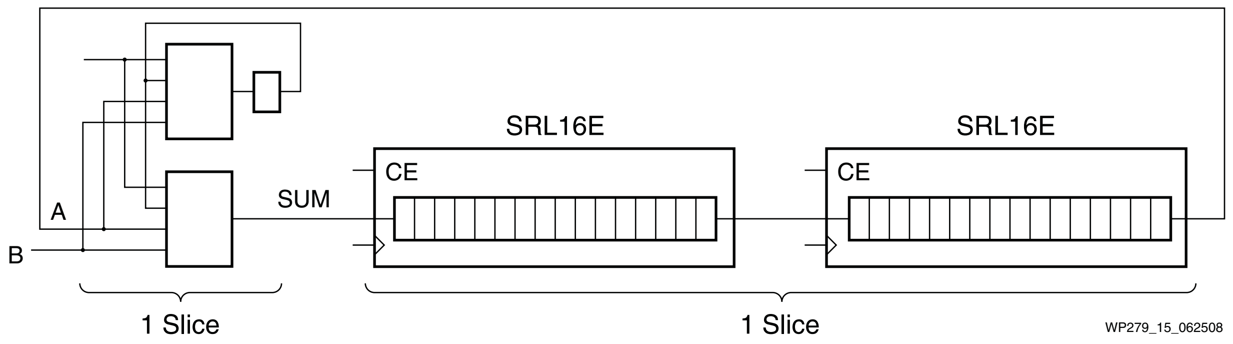

Figure 15: **32-Bit Accumulator Comprising Two Slices**

Back-to-Front Data

The serial adder and subtracter functions work with the LSB first; however, as shown by the communications with the TLC320AC01C, serial samples can be derived with the MSB first (see Figure 16). The order of the data must be changed, and this again is ideally suited to the SRL16E primitive.

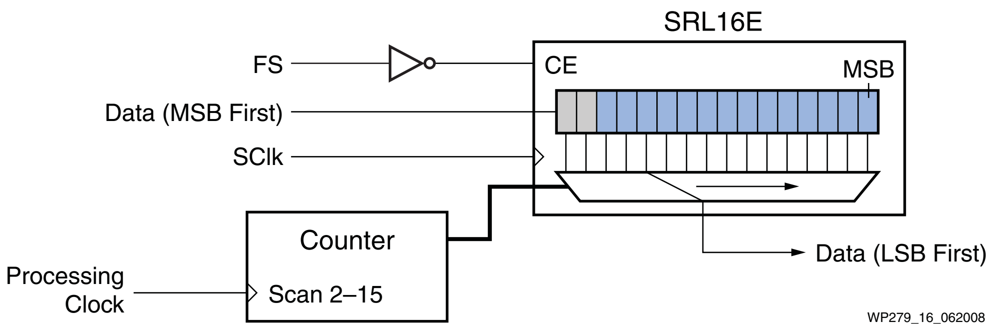

Figure 16: **Bit Reversal Using SRL16E**

The SRL16E also performs a conversion between clock rates. The serial data clock can be applied to the SRL16E directly with no requirements for a clock buffer. This is because all the flip-flops of the shift register are contained in the same look-up table and share a common local clock with zero skew. When the frame strobe is active (Low), data is enabled to shift into the SRL16E with the MSB first. After the 16-bit transfer is completed, the data remains static and can then be read via the embedded multiplexer, which is a combinational process. A counter can select the required sample bits and, of course, read them LSB first.

The key to serial processing is to ensure that each adder and subtracter uses the correct bits during each clock cycle. In this implementation, the accumulator addition process starts with the least significant bit of the fraction part of the DC offset, but the subtraction must start with the least significant bit of the integer part of the DC offset. This is easily solved by splitting the accumulator storage into two shift register delays such that a tapping point is achieved at the LSB of the integer storage point.

In this implementation, the input samples are assumed to be 14 bits long. This means that the serial subtraction process takes 14 clock cycles. However, the accumulation process must be 22 bits, because it includes the 8-bit fraction, and therefore requires 22 clock cycles. To apply the 14-bit result of subtraction to the 22-bit accumulation process, sign extension must be performed again. In the serial domain, the sign extension is easily achieved by holding the MSB generated by the subtracter static in a flip-flop by deasserting a clock enable at the end of the 14th clock cycle (see Figure 17).

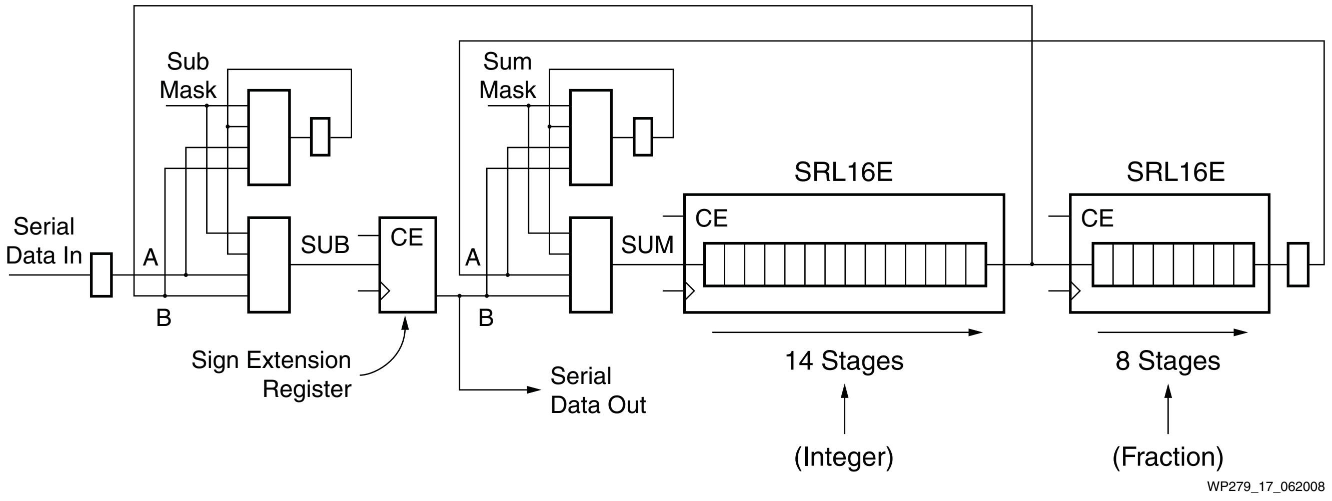

Figure 17: **Serial Implementation of DC Offset Removal**

State Machine Made Easy

The state machine used to control each event is also simplified by the use of SRL16E delays. This can be a form of a one-hot state machine that is allowed to become cold between the bursts of activity required to process each new data sample. The hot state is injected in the form of a single clock cycle pulse that is applied coincident with the LSB of the serialized 14-bit data sample.

Simple flip-flops delay this initial start pulse and ensure that the serial subtracter and serial adder have the carry mask applied coincident with processing the LSB in each case. The SRL16E components are used to delay the initial pulse for 14 and 22 clock cycles to control the duration of the serial subtract and serial accumulation processes. In each case, a flip-flop is set by the initial start pulse and enables the process to begin. When the pulse emerges from the SRL16E, it is used to reset the flip-flop and, hence, stop the serial processing (see Figure 18).

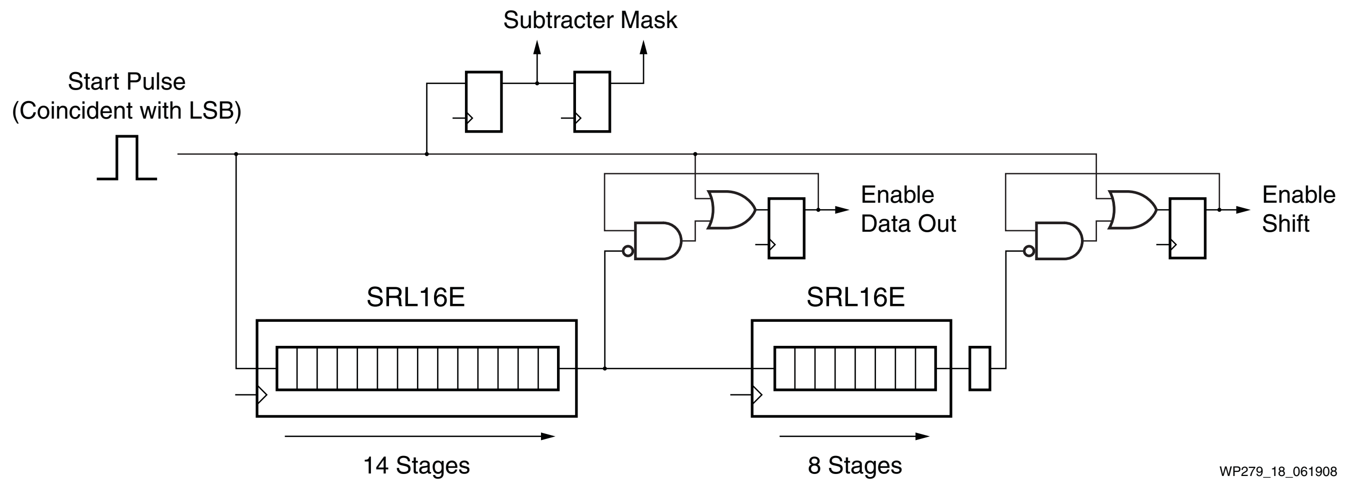

Figure 18: **Control Signals for Serial Implementation of DC Offset Removal**

Conclusion

The first half of this white paper discusses how DSP can be used in a practical manner. A simple analog circuit can be represented in the digital domain to create a very efficient and practical function. The accumulator, subtracters, and multiplier are all able to operate in excess of 100 MHz in all devices. Therefore, it is also possible to create a high-performance circuit that is able to take samples directly from the majority of high-performance ADCs.

The second half of this white paper shows how a careful consideration of coefficient values can help to remove real multiplier logic and thereby significantly reduce the size of a function. Serial processing can further reduce the size of an implementation for lower sample-rate applications. In this type of application, the data samples are most likely provided in a serial format, and thus the requirement to convert to a parallel format for processing is eliminated.

The SRL16E is used in dynamic addressing mode to act as a very efficient bit reordering circuit as well as to provide pure delay. Serial processing can be quite difficult to implement. However, the SRL16E's ability to provide a complementary state machine with direct control over the scheduling of events has made serial processing much easier and smaller than it has been in the past using counters. The serial implementation requires only six slices to implement the 14-bit serial DC offset removal circuit and even the smallest FPGA can support 32 or more channels.

For an 8 KHz sample rate, this serial process only requires a minimum clock rate of 176 KHz. Given that any Xilinx FPGA can easily support 100 MHz clock rates, just one serial processing circuit could actually support 568 channels using time division multiplexing (TDM) techniques. In this situation, the amount of memory required to store the DC levels (accumulator values) would be better supported by block RAMs acting as cyclic buffers.

References

The following reference provides additional information useful to this document:

1. Texas Instruments TLC320AC01C Data Manual
http://focus.ti.com/lit/ds/symlink/tlc320ac01.pdf.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
07/18/08	1.0	Initial Xilinx release. Based on two previously published TechXclusives articles by the same author.

Notice of Disclaimer