WP271 (v1.0) May 22, 2008

Saving Costs with the SRL16E

By: Ken Chapman

Although the SRL16E 16-bit shift register sounds like an additional component, it is an alternate operating mode for the Look-Up Tables in which they become 16-bit shift registers. While a synthesis package will automatically apply the SRL16E in some situations, substantial benefits can be realized by understanding the potential of the SRL16E and deliberately applying it to your designs.

This white paper provides examples to help your understanding of the capabilities and use of the SRL16E to improve the performance and lower the cost of your designs by as much as an order of magnitude.

© 2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Modes of Operation

The SRL16E was introduced in the Virtex® FPGA architecture and is included in all variants of the Spartan®-3 family. It is an alternate operating mode of a Look-Up Table in which it becomes a 16-bit shift register as shown in Figure 1. The added capability this configuration provides can lead to dramatic improvements in performance and cost savings to your design. However, the operation of this mode is not obvious, and an understanding of the underlying structure of an SRL16E is necessary to maximize the benefits available from this configuration. The information and examples contained in this White Paper will help you gain this understanding.

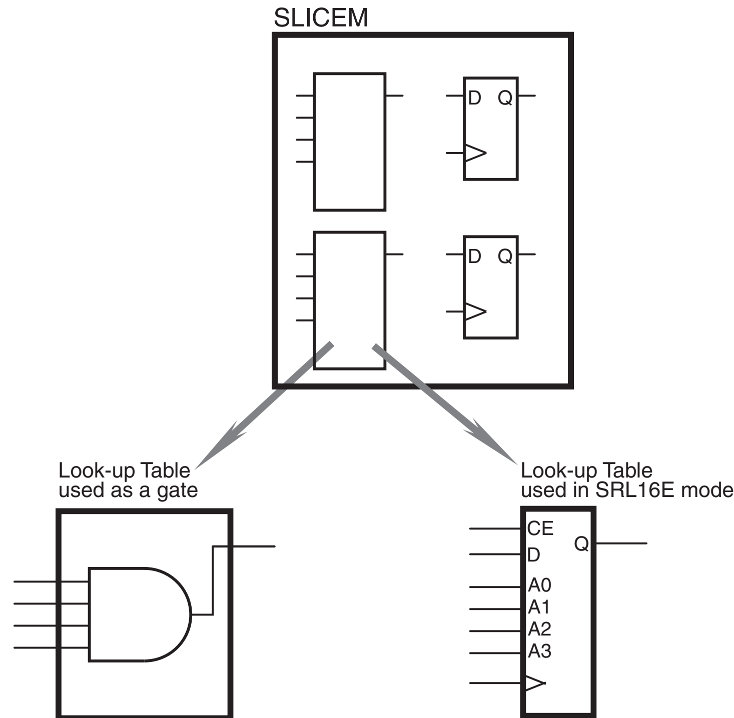

Figure 1: **A SLICEM Look-Up Table Used as a Gate or Shift Register**

Look-Up Table as an AND Gate

Figure 2 shows a Look-Up Table configured as a 4-input AND gate. The select lines on the 16 to 1 multiplexer are used as input signals *I3*, *I2*, *I1*, and *I0*. The multiplexer output *O* will serve as the AND gate output. The data value for each multiplexer input are the 16 configuration bits which are set via the configuration bit-stream. Only when all four multiplexer select lines *I[3:0]* are logic one is the configuration cell containing logic one selected. In any other case, the multiplexer will select a cell containing a logic zero.

The INIT parameter is a description of the bit pattern stored in the configuration cells. This hexadecimal value representing the bit pattern 1000 0000 0000 0000 (8000 hexadecimal) can be applied manually to a Look-Up Table primitive (LUT4 component), or is more normally generated automatically by an HDL synthesis package.

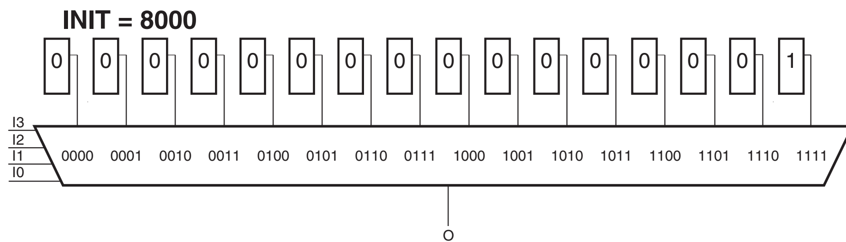

Figure 2: **Look-Up Table Used as a 4-input AND gate**

Look-Up Table in SRL16E Mode

When the Look-Up Table is used in SRL16E mode, the basic structure is similar, but now the 16 configuration cells are organized as a shift register connected to the multiplexer as shown in Figure 3. The 4 select line inputs to the multiplexer are labeled *A3*, *A2*, *A1*, and *A0*, and their ability to select one of the 16 inputs remains the same. In this mode, the multiplexer inputs can still be initialized via the configuration bit-stream (using an INIT value), but the contents can also be modified by shifting in new data on the *D* input.

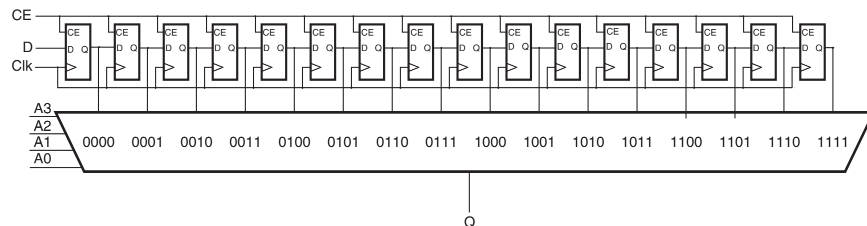

Figure 3: **Look-Up Table in SRL16E Mode**

The output is labeled *Q* to reflect that the selected value is a *Q* output from one of the shift register flip-flops. Note that the shift register output delay is determined by which flip-flop output is selected by the *A[3:0]* inputs, and can be from 1 to 16 clock cycles. This selection is also independent of the clock and clock enable inputs.

Basic SRL16E Applications

The use of the SRL16E can facilitate space-efficient designs and improved performance by increasing the potential device gate count. Each SLICEM contains 34 flip-flops [2 x (16+1)], which greatly increases the gate count of the device. The 50,000-gate XC3S50A Spartan-3A FPGA contains 704 slices, half of which (352 slices) are SLICEM and could implement 11968 bits of shift register. This is the equivalent to about 72000 ASIC gates (without even considering the 54kbit of block RAM).

Simple Shift Register

The most basic use of the SRL16E is as a shift register or digital delay. For example, a 10 clock cycle delay requires 10 flip-flops connected as shown in Figure 4.

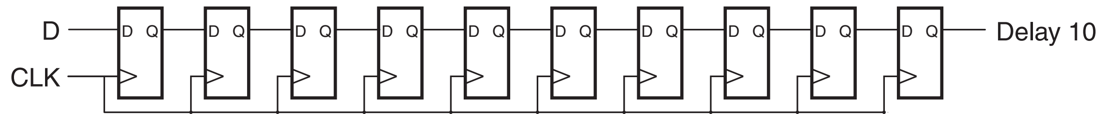

Figure 4: **10-bit Shift Register**

One way to implement this function is to connect ten of the individual flip-flops available on each slice. Since two flip-flops are available on each slice, this would occupy 5 slices of the Xilinx Spartan-3 generation FPGA architecture. However, a more efficient implementation can be created using one of the two SRL16Es available on a single SLICEM.

The 10 clock cycle delay is available at the output of the tenth flip-flop. A3, A2, A1, and A0 can be tied to Vcc and GND as shown in Figure 5 to a value of 1001 which will select the output of the tenth flip-flop (note that the addressing of the flip-flop outputs start at 0000, so the tenth flip-flop has address 1001). In this example, the shift register is permanently enabled; hence, the CE input is also tied High to Vcc.

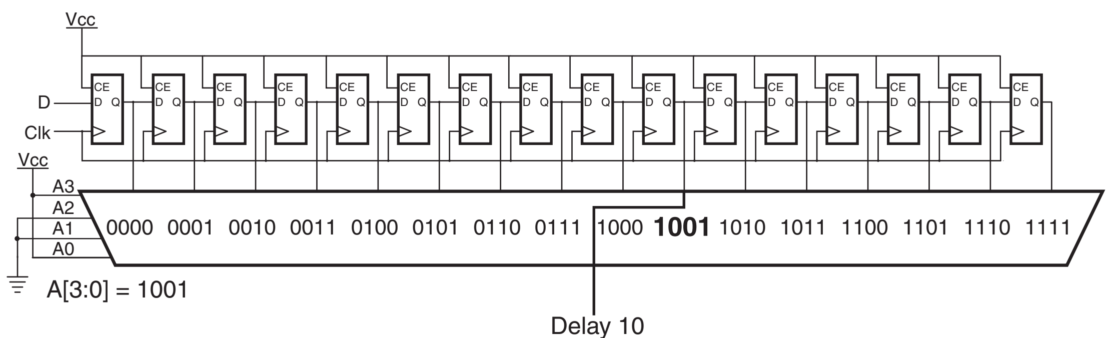

Figure 5: **10-bit Shift Register Using the SRL16E**

It is obvious the 16 flip-flops and 16:1 multiplexer comprising a SRL16E permit it to support a delay from 1 to 16 clock cycles. Less obvious is that high performance is easier to achieve in a system design. Since all the flip-flops are fixed in the silicon, the connections between them are short and predictable. The lower utilization of the device reduces power consumption and also enables other logic to be placed in closer proximity. Although Vcc and GND signals must be routed to the SRL16E, these static signals will have no impact on timing or power consumption.

Design Considerations

Use of the SRL16E in a delay configuration is relatively simple. Being aware of when it can be used is vital in ensuring maximum utilization of a device. This is particularly true when using an HDL synthesis tool. Believe it or not, you may already have used this powerful feature because most synthesis tools will automatically insert SRL16E into your design to replace multi-stage delays. The issue is that failure to use the SRL16E either manually or automatically will have a large impact on the area of silicon occupied. The following points are worth considering when designing with any tool.

No Reset

Although the SRL16E supports a clock enable (CE) for each of the 16 flip-flops, it does not provide a reset. When you design using schematics, you must deliberately select the SRL16E in order to use this mode. When you design using HDL where the SRL16E mode is automatically selected, you must be careful not to imply a reset on elements forming a shift register delay. Note that all flip-flops including those in each SRL16E are reset by the configuration process and hence start in a known condition. HDL coding styles and templates often include a reset to achieve this same condition in an ASIC and remove unknown states from a simulation. Removing these unnecessary resets from code can allow many more flip-flops to be automatically consolidated into the SRL16E. As a result, designs can shrink by more than 10 percent, often enabling a smaller device at reduced cost (see WP272).

Simulation Confusion

Simulation of delays should be simple. However, it appears to confuse everyone at some stage and therefore a small explanation is worthwhile. In this example the delay is 6 clock cycles which would relate to the A[3:0] input being forced to '0101'= 5. The confusion can occur because the data appears after just 5 clock cycles (Figure 6).

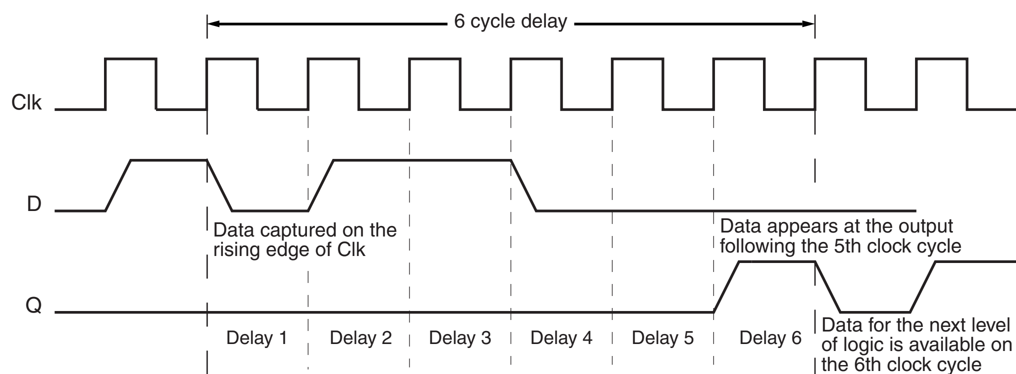

Figure 6: **6 Cycle Delay**

Delay Examples

Consider some applications in which shift register delay can be used, or indeed occurs by default.

Pipeline Compensation

For high performance designs, Xilinx FPGAs provide excellent results when fully pipelined. Unlike an ASIC, the flip-flops in each slice are already provided and waiting to be used. Although the pipeline registers are essentially free, a pipelined system may have additional cost resulting from pipeline compensation in other paths. Figure 7 shows the addition of 9 values of 16-bits. It can be seen that the 9th input has to be delayed by 3 cycles to compensate for the addition tree of the other 8 inputs. This compensation delay requires 48 flip-flops (3x16) which would occupy 24 slices. The SRL16Es reduce this to just 16 Look-Up Tables in 8 slices.

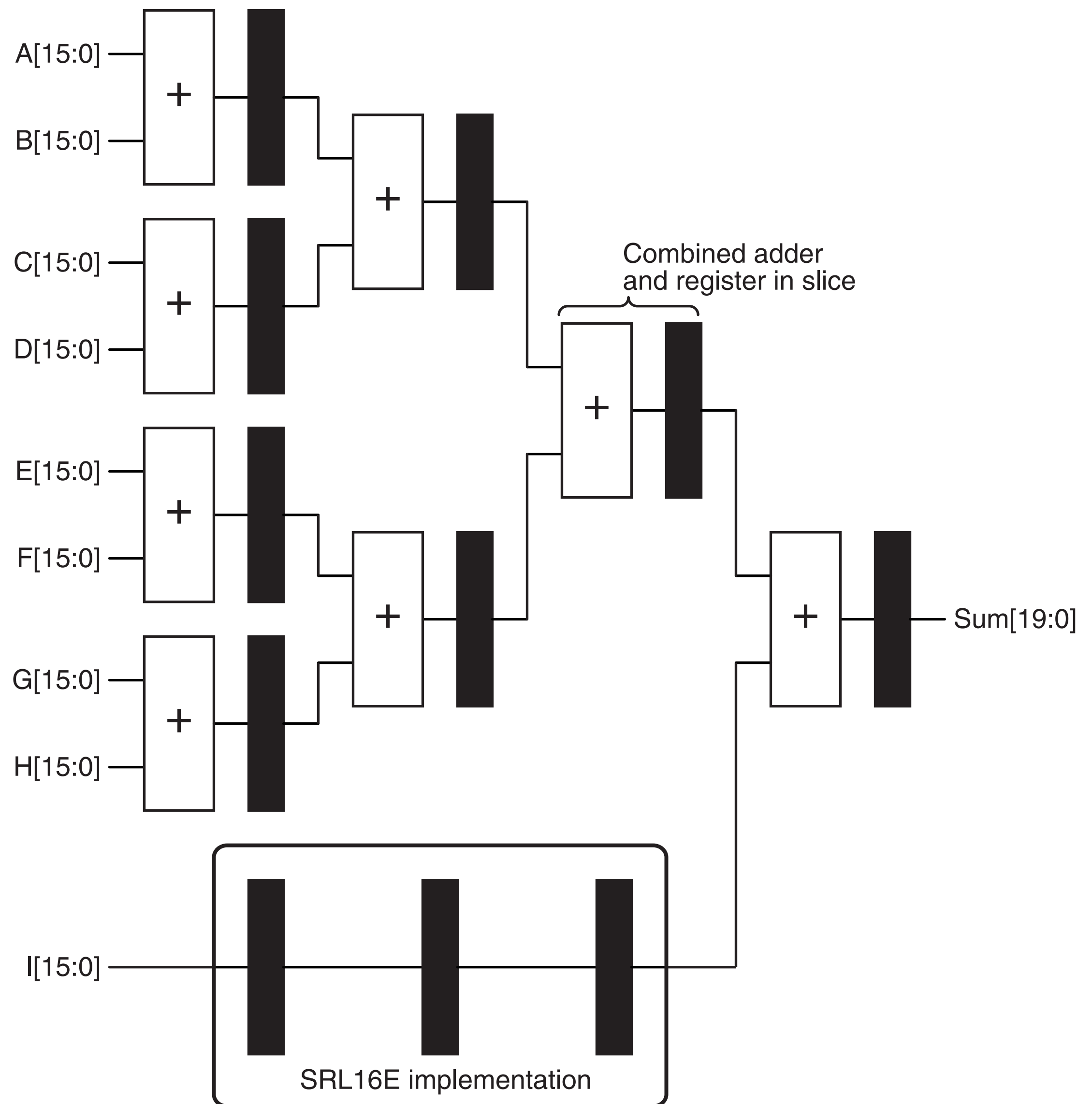

Figure 7: **Using SRL16E for Pipelining**

Pseudo Random Noise Generator

A pseudo-random noise generator (PN) is based on a *Linear Feedback Shift Register* (LFSR) counter technique.

There are two styles of implementation, but both make use of simple shift register delay. The PN generator is responsible for producing a pseudo-random signal that produces logic zero and logic one levels. Note that a pseudo-random number generator is the same structure, but would observe all bits of the shift register as a parallel word. This would not enable the SRL16E to be used other than to provide a single flip-flop.

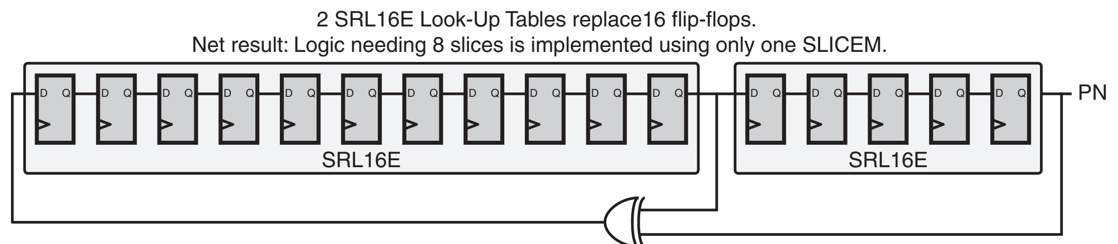

Figure 8: **Using SRL16E for Linear Feedback Shift Registers**

In some applications the realization that the SRL16E can provide such efficient shift register delays can be utilized to greatly simplify designs. This is explored more in the "Pulse Generation and Clock Division" topic.

Serial Frame Synchronizer

This is a common application seen in telecommunications and networks. Data is passed serially between units (for example at 2.048 Mbps) and is formatted into frames or packets. In order to synchronize to these packets, a start and end code is included. Although the start code should be unique, it is possible that the data itself may contain the same pattern. To reduce the probability of incorrect synchronization, the end of frame code can be used to further qualify the data. Only when the start code and end code occur at the correct times is the data accepted. Using a state machine approach to solve this design problem, the start code pattern is identified and then the data stream is stored in a buffer. Data bits are counted, and at the expected count, it would then check for the end of frame code. If this code does not occur when expected, or if a start code occurs beforehand, then the buffer is overwritten with new data. This approach still has the possibility of losing data.

The ideal solution is to hold an entire frame in a shift register and test for the start and end patterns at *every* bit position. Figure 9 shows this approach implemented as a 512-bit frame detector. One frame consists of a serial stream containing a 16-bit start code, 480 bits of data, and a 16-bit end code. As each new bit is shifted in to the *Serial In* input, data in the shift register advances by one position. The 16-bit AND gates connected to the flip-flop outputs at the front and the back of the 512-bit shift registers will detect the simultaneous occurrence of the start and end codes and provide a *Sync* pulse to indicate the data contained in the middle 480 flip-flops is valid. If the flip-flop pair contained in each slice was used to build this 480-bit shift register, it would occupy 240 slices and would be considered expensive. A more efficient approach will

result by using the 16 flip-flops available in the SRL16E mode. This will reduce the 480 bit shift register to 30 SRL16Es in 15 slices. In this way the entire synchronizer can be realized in just 31 slices, resulting in an equivalent gate count of over 3000 achieved in less than *5 percent* of the Spartan-3A XC3S50A (50,000 gate device).

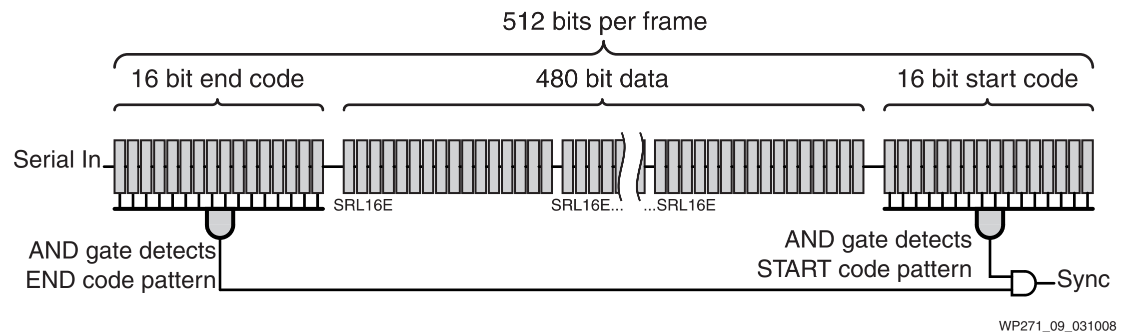

Figure 9: **Using SRL16Es as a Serial Frame Synchronizer**

Intermediate SRL16E Applications

The next SRL16E applications require additional thought about the alternative ways in which a problem can be solved. It is often possible to find a solution to a problem using shift register delay. Although HDL tools will automatically convert simple delays into SRL16E primitives, they are not able to change the structure of the design. As the engineer, you are able to discover these fresh approaches. The following two examples illustrate some realistic cases of such structures.

Running Average Using an Adder Tree

The running average (or more accurately the running sum) of a set of samples is a simple FIR filter having a smoothing or low-pass response. In Figure 10, the output is the sum of the last 8 samples held in a shift register. If we assume 16-bit wide input word, then we need 8 x 16-bits = 128 flip-flops to form the shift registers plus an adder tree made of 4 x 17-bits + 2 x 18-bits + 1 x 19-bits = 123 bits. Hence, this implementation requires 128 flip-flops for shift registers, 123 bits of addition, and would require 126 slices to implement.

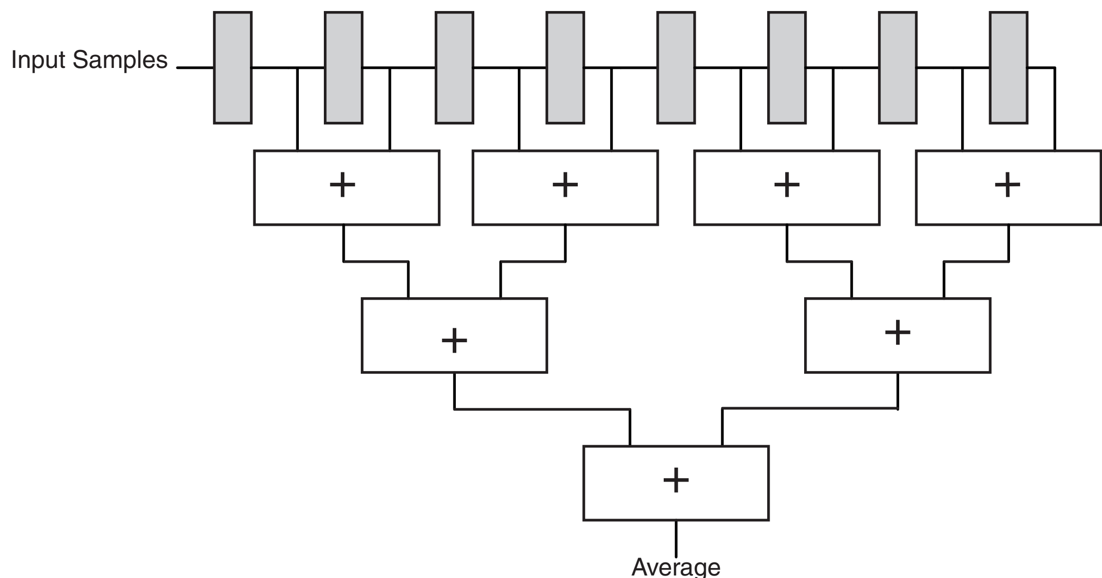

Figure 10: **Running Average Using Non-Pipelined Adder Tree**

Assuming that the non-pipelined adder tree shown in Figure 10 can provide the necessary performance, the adders and registers can be mapped into the same slices, reducing the size to 64 slices. This substantially reduces space requirements. However, using knowledge of SRL16E, and the characteristics of a running average function, we can go much further.

Running Average Using an Accumulator

In order to reduce complexity, observe that for this running average function a sample entering the shift register contributes to the result for 8 clock cycles. Although a sample always contributes the same value to the result, it is doing so via different paths through the addition tree each cycle.

Figure 11 shows an alternative solution that uses this fact by adding the new sample to an accumulator that remembers the sample's contribution. After 8 clock cycles, we then subtract the delayed sample from the accumulator. Not only does this reduce the adder tree to a simple subtractor and accumulator, but it also puts the shift register into the form of a simple delay line which is easily absorbed into SRL16E primitives. For this example, the 16-bit input samples still require 8 x 16 flip-flops for the shift register, but now fit into just 8 slices. The subtractor is 17-bits and the final accumulator is 19-bits (requiring a further 18 slices). The total size is 26 slices, a *60 percent* reduction in size when compared to the original implementation. Also note that as the length of the running average is increased, the savings are significantly greater.

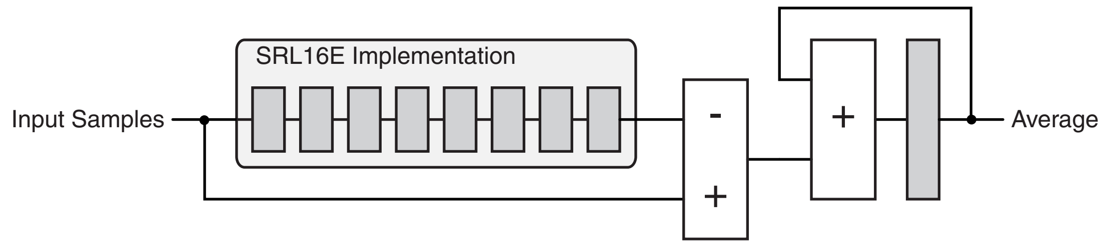

Figure 11: **Running Average Using Accumulator**

Pulse Generation and Clock Division

Unless there are special reasons not to do so, the use of a single clock source (distributed via one of the global clock networks) should be used for all elements of a design. Clock enable pulses should then be used to enable system subsections to operate at a lower rate. In Figure 12, the enable pulses are used to enable a counter. Clearly, the pulse must occur at regular intervals and have a duration of one clock cycle.

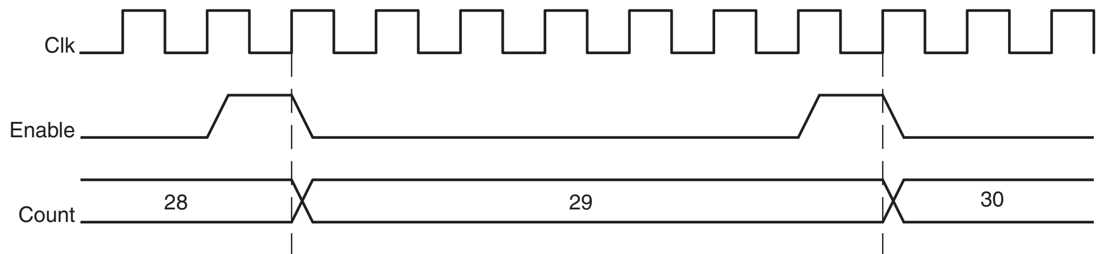

Figure 12: **Pulse Generation**

Figure 13 shows a practical example of an RS-232 receiver module which receives serial data at 9600 bits per second. Typical of most RS-232 interfaces, this unit uses a timing reference which is 16 times the bit rate. In line with good design practice, this macro takes this reference as a clock enable pulse.

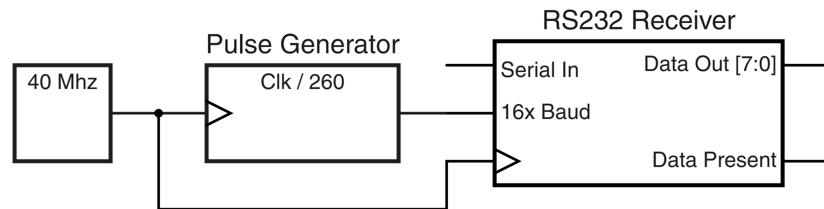

Figure 13: **Pulse Generation for RS-232**

Pulses at 16 x 9600 bits per second (153,600 kHz) are not an exact integer division of 40 MHz. However a division by 260 is equivalent to 16 x 9615 which is adequate for RS-232. This means that a pulse must be active for one cycle in every 260. One way to achieve this is by using a 9-bit counter with an AND gate to detect a count value 259. The output of the AND gate provides the pulse and synchronously resets the counter (Figure 14). This solution requires 6 slices of logic.

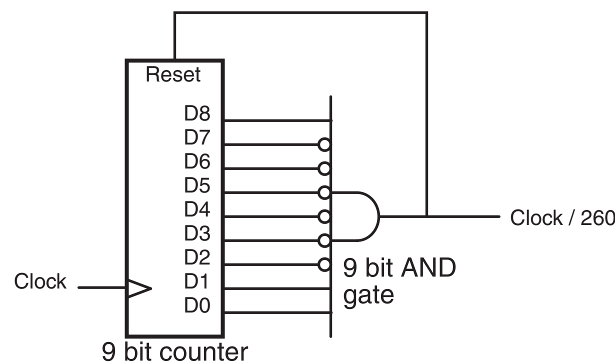

Figure 14: **Counter-Based Pulse Generator**

A counter can be viewed as a simple state machine. In this case, the 9 bits provide a possible 512 states of which the first 260 are actually used. Each state leads to the next until reaching the 259th state, when it returns to zero. An equivalent *one-hot* state machine is one in which a different shift register flip-flop is used to represent each state. The active state is at logic one; all other states are logic zero.

A one-hot state machine made by combining multiple SRL16Es is very practical. Figure 15 shows that 16 SRL16E primitives and 4 flip-flops can provide the 260 states required. This solution is larger than the counter-based pulse generator, but the simplicity of this technique makes it practical.

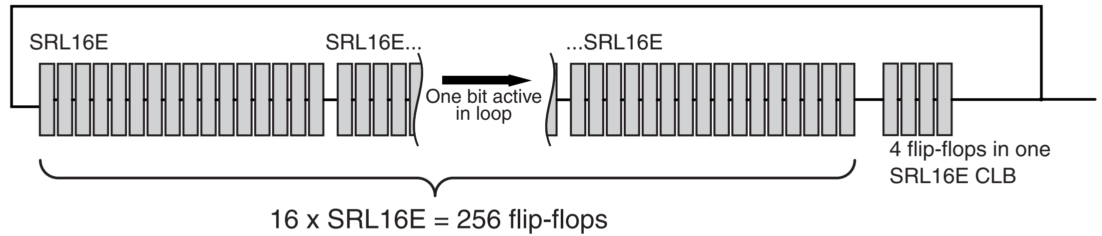

Figure 15: **Using SRL16E for One-hot Counter**

Multi-stage Dividers

Although the one-hot state machine approach is ideal for smaller division factors, at some point the number of flip-flops required becomes impractical. A solution to this is to use multiple stages to divide and conquer.

The division by 260 can be broken into 2 stages (Figure 16). The first stage divides the clock by 26 and provides enable pulses to the second stage. The second stage divides the enable pulses by 10.

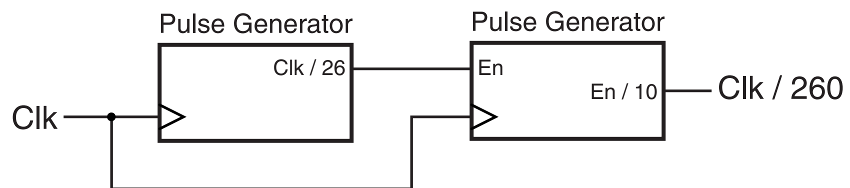

Figure 16: **Dividing Pulse Generator into Two Stages**

The first stage requires a 26-state looped delay around which a single hot bit is clocked to produce the enable pulse. This can be created with 26 flip-flops contained in just two SRL16E primitives in a single slice (Figure 17). This is a more efficient implementation than the equivalent function created using a 5-bit counter and decoder which requires 3 slices.

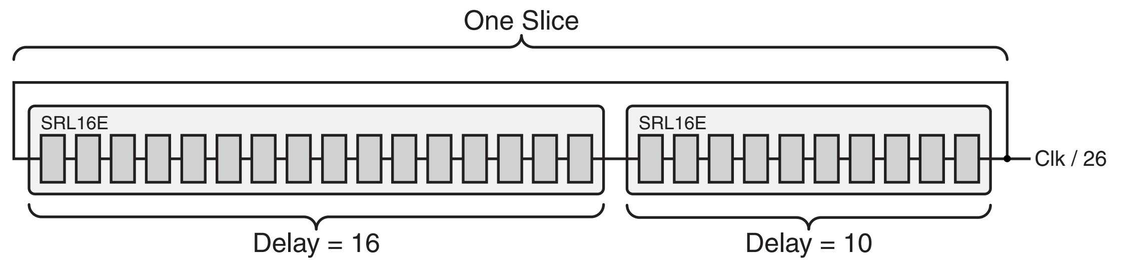

Figure 17: **Two Stage Divide by 260**

The second stage differs from the first stage since it must divide the clock enable pulses from the first stage, rather than the clock directly (Figure 18). For this reason, the clock enable input is driven.

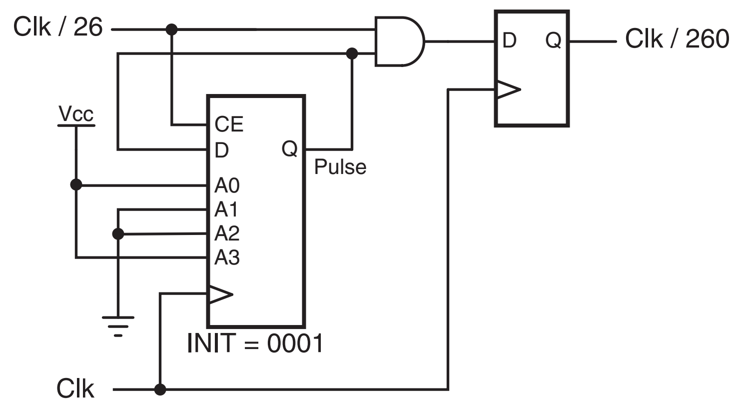

Figure 18: **Using SRL16E for Second Stage of Pulse Generator**

Since the SRL16E is only enabled once every 26 clock cycles, the output pulse is active High for 26 clock cycles (Figure 19). To generate a single cycle clock enable pulse, this long pulse is gated with the input enable. The flip-flop ensures high performance, as such an enable pulse may have a high fan-out.

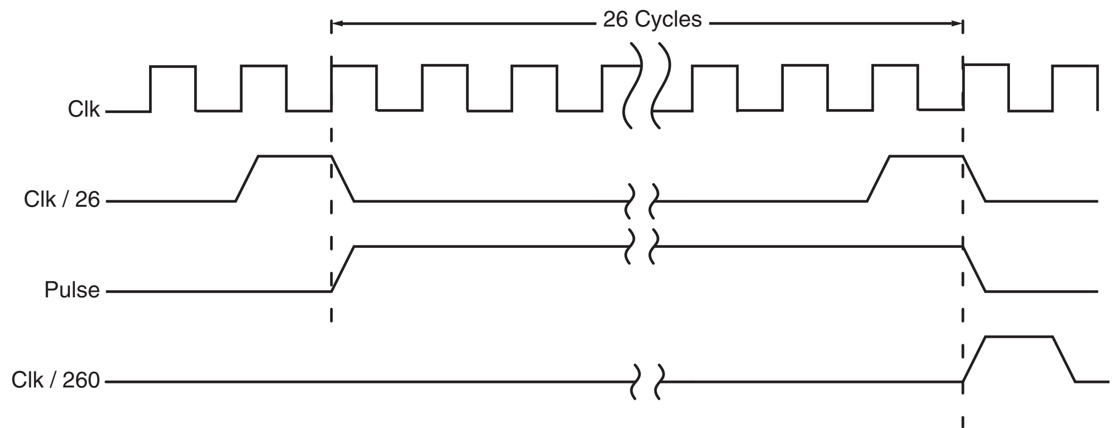

Figure 19: **Timing of the Divide By 260 Implementation**

Forcing the Hot State

Because this kind of pulse generator works by recycling the hot (logic one) state, the hot state must be present in the delay loop at the start of operation. There are two ways to achieve this.

Using INIT

One method is to set the hexadecimal INIT value on the SRL16E Look-Up Table (Figure 20). If necessary, the position of the hot bit can be determined during the start-up sequence. Note that the hot bit must be defined before the delay tapping point. If INIT is set to 0001, one loop of the delay is required before the first pulse is seen at the output.

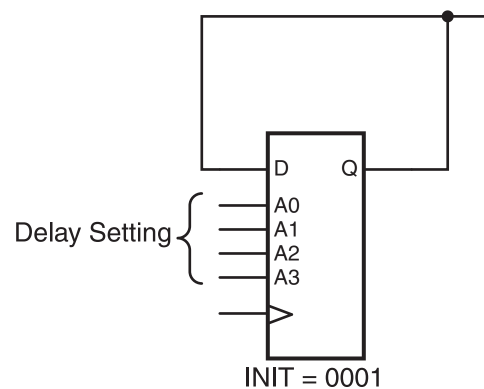

Figure 20: **Initializing Hot State using the INIT Value**

Using a Flip-flop

Although the INIT parameter provides total control, this may not be easy to achieve via synthesis unless component instantiation is used. In this case, the default INIT=0000 can be used and the hot bit injected using a flip-flop (Figure 21). This flip-flop can be initialized with logic one by declaring (or inferring) an asynchronous preset. Obviously, the delay formed in the SRL16E is now one less. The flip-flop has the advantages of increasing the maximum delay available by one and improving the clock-to-output performance of the pulse generator.

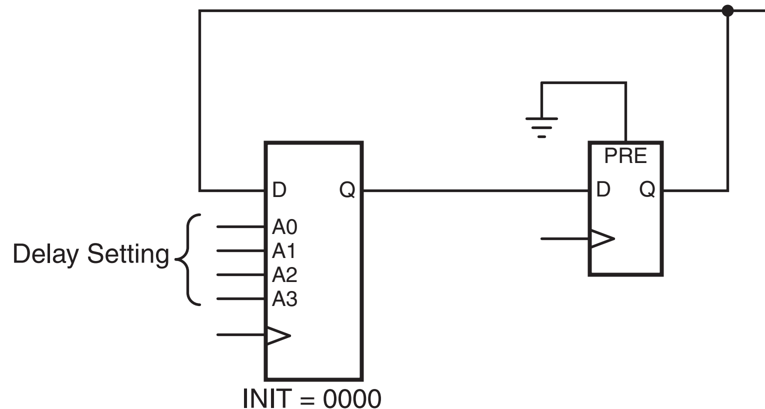

Figure 21: **Initializing Hot State using a Flip-Flop**

Pattern Generation

Since the SRL16E can be pre-initialized with any INIT value, it can be used to produce a repeatable pattern. In Figure 22, a clock signal is generated that is 400 times less than the input clock. In this example, the object is to produce an output waveform having approximately a 50 percent duty cycle. The solution uses a divide by 16 stage which enables a divide by 25 stage. Note how the INIT values set the pattern of the output waveform.

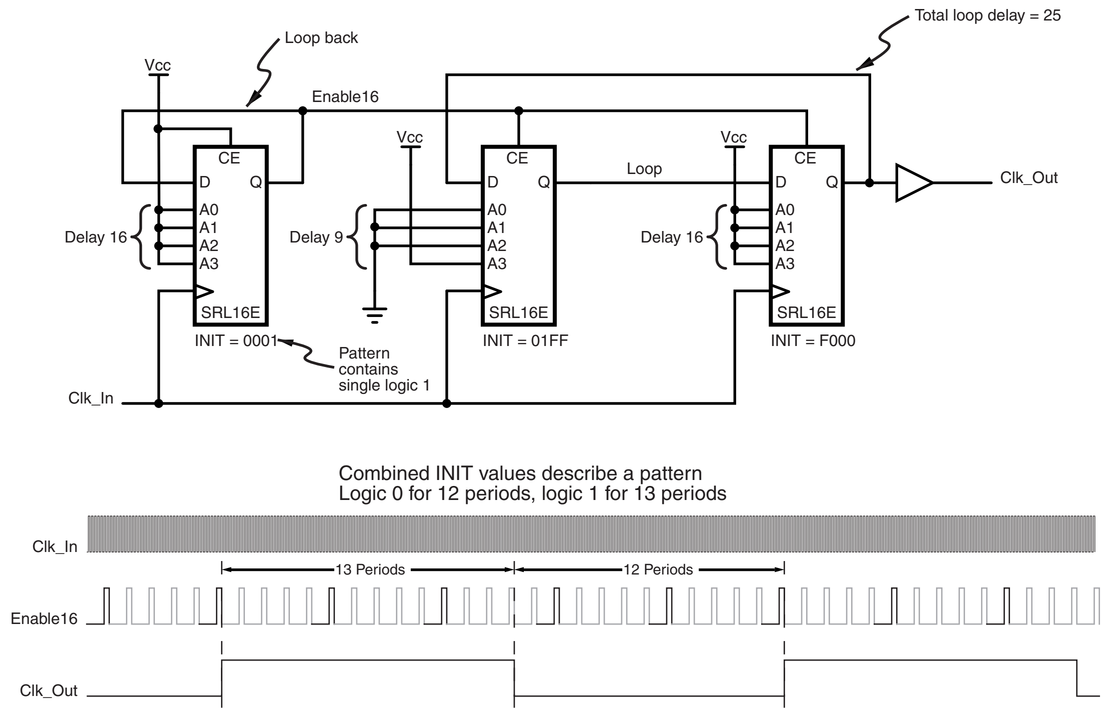

Figure 22: **Using Init Values to Define a Pattern**

Very High Performance Clock Division

The example shown in Figure 22 can be implemented in 2 slices. Because all logic in a single slice shares a common clock, there are no clock skew issues between the elements; thus, the input clock does not need to be provided from the global clock resource. The close proximity of the elements also yields very high performance (in excess of 350 MHz). Such a circuit may be useful in dividing a high speed clock before use of a global clock resource in order to minimize the power dissipation. In general, a single clock and enable pulse is preferable.

Complex State Machines

We have seen that the simple clock division or pulse generator is a form of one-hot state machine. This technique can be expanded to form more complex state machines. The use of shift registers to delay control pulses is an effective implementation method that can be easier to design than you might think.

RS-232 Receiver

In this example, a simple RS-232 receiver is considered. Figure 23 shows a serial data word provided at a rate of 9600 bits per second with a single start bit (active Low) and single stop bit (active High). Although the serial data arrives asynchronously, the basic timing of the data is implied from the bit rate.

A reference time signal is provided at 16 x bit rate. The falling edge of the start bit is used to trigger a state machine which uses the timing reference to approximately locate the mid-position of each data bit.

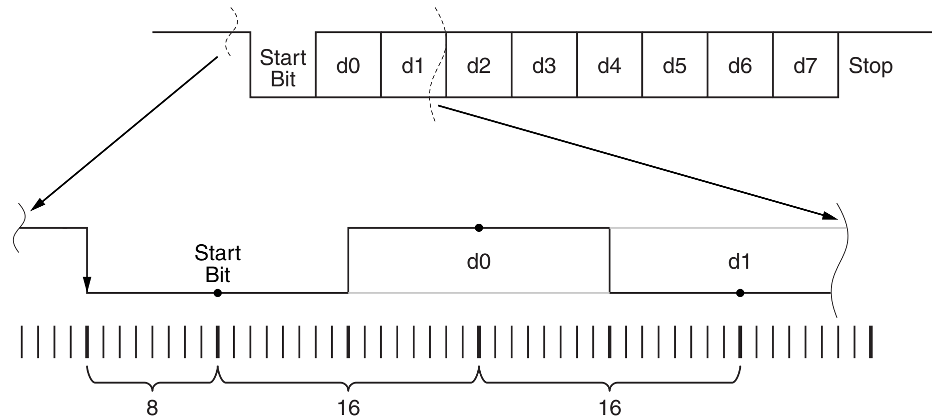

Figure 23: **RS-232 Serial Data Waveform**

Figure 23 shows that the state machine must first count 8 timing reference periods to locate the mid-position of the start bit, and then count every 16 timing reference periods to locate the midpoint of each data bit and the stop bit. Once the stop bit has been sampled, the state machine must return to the idle condition in which it locates the next falling edge of a start bit.

Although this example does not show all the details required by a UART state machine, it does illustrate the main functionality and the way in which the SRL16E primitives are used. Figure 24 shows a 24 state one-hot state machine with each state represented by the flip-flops contained within two SRL16E elements. All states are Low until a High pulse is injected when a High to Low transition occurs at the RS-232 input. While not shown in this example, a full implementation must prevent multiple hot states from being generated on subsequent data transitions.

The first SRL16E delays this pulse until the midpoint of the start bit, and then injects the hot state into the second SRL16E. This will circulate every 16 bit rate-enabled periods to locate the midpoint of each data bit. At the end of the data transmission, the hot state is prevented from re-entering the SRL16E, and the state machine returns to being inactive. A complete RS-232 receiver for this specification can be made in 8 slices.

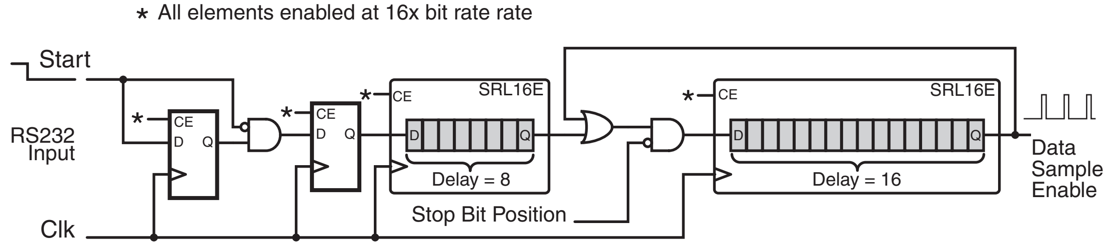

Figure 24: **RS-232 Receiver Logic**

Advanced SRL16E Applications

In the next examples, we will move away from using the SRL16E to implement a fixed delay. Instead, the multiplexer select lines will be dynamically addressed.

FIR Filter

The most important fact regarding this mode of operation is that the shift register and the multiplexer are totally independent. The shift register takes data from the *D* input under the control of the clock (*CLK*) and clock enable (*CE*) signals. The output at *Q* depends only on the *A[3:0]* inputs. Although it is tempting to believe that a pin called *Q* is associated with a synchronous clocked output, it must be understood that the multiplexer operation is completely combinatorial and has nothing to do with the clock. The use of the label *Q* indicates the multiplexer is selecting one of the 16 shift register flip-flop outputs.

For a synchronous output, the associated flip-flop within the slice can be used. Good design practice requires this flip-flop to use the same clock as the SRL16E.

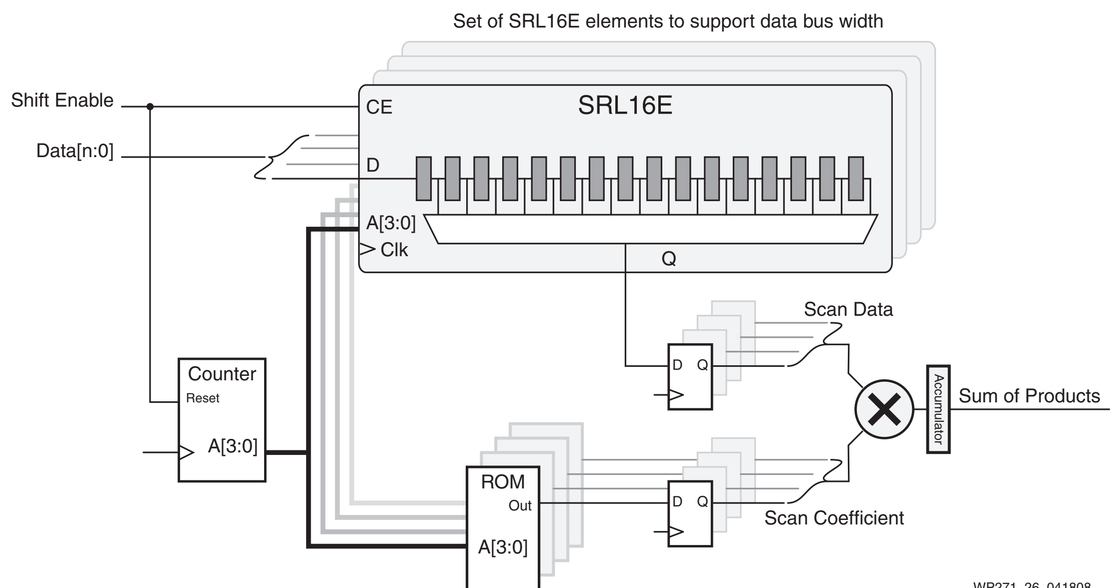

Figure 25: **FIR Filter Logic**

In this example the SRL16E elements form the taps of the FIR filter. A new sample causes all previous samples to move along the shift register and the oldest sample will be discarded. To calculate the filter output a single multiply and accumulate unit is used over 16 clock cycles.

With the data held static in the shift register (CE is Low), a counter addresses the multiplexer within the SRL16E to select each tap in turn and apply the contents to the multiplier. The same address is used to select the appropriate coefficient from a ROM. In the complete implementation the accumulator would be reset at the beginning of each calculation.

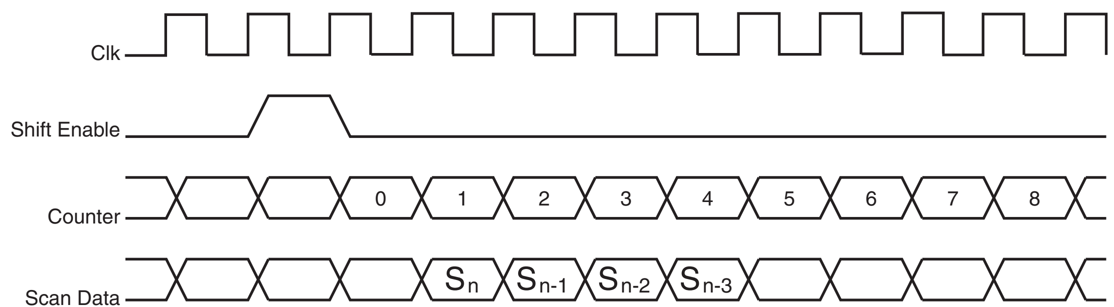

Figure 26: **FIR Filter Timing**

FIFO

A FIFO is normally based on a dual port memory (for simultaneous write and read operations), a pair of address counters (for write and read pointers) and control logic to detect full or empty conditions. The SRL16E can be used to form a FIFO as shown in Figure 27 through Figure 30. This implementation has the advantage of greatly simplified counters and control logic and twice the density of dual port RAM.

Operation

1. This example begins when 5 data words have been written to the FIFO. The counter is at value 4 to select the 5th tapping point and hence the oldest data is available at the output.

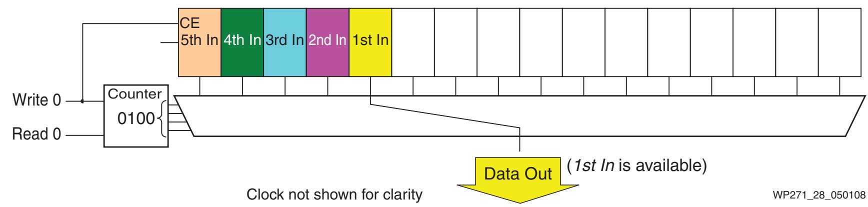

Figure 27: **SRL16E After Writing Five Words**

2. *Data WRITE.* A new data write causes all data in the shift register to advance by one position. Note the counter has been incremented and so the multiplexer is still selecting the oldest data to be presented to the output.

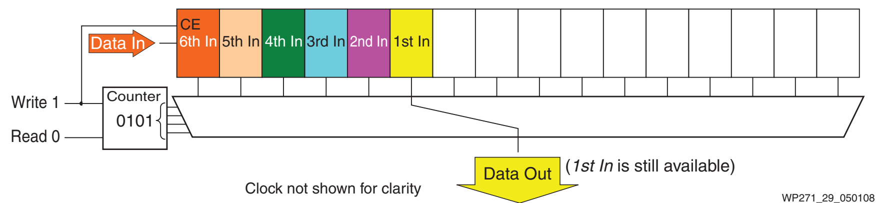

Figure 28: **SRL16E After Writing a 6th Data Sample**

3. *Data READ.* Reading data does not affect the contents of the shift register. Note the counter has been decremented so the multiplexer is now selecting the next oldest data to be presented at the output.

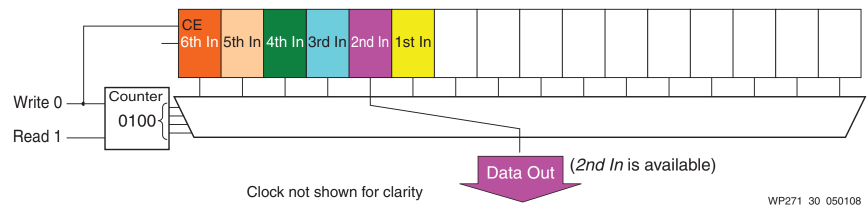

Figure 29: **SRL16E After Data Read**

4. *Simultaneous data WRITE and data READ.* The shift register has advanced by one position but the counter remains at the same value. As a result, the next oldest data moves into the selected position. The previously read data is still contained in the shift register but is no longer needed and will eventually be lost completely.

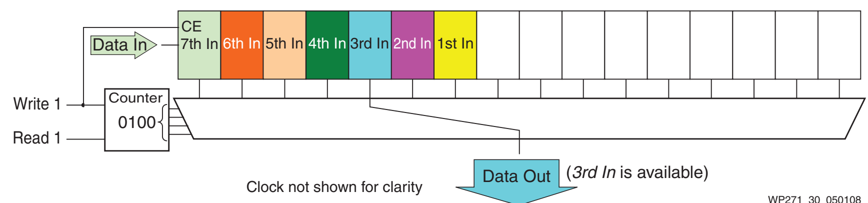

Figure 30: **SRL16E After Simultaneous Data Write and Data Read**

Complete RS-232 Receiver

Figure 31 illustrates a complete RS-232 receiver with baud rate generator and FIFO data buffer. This example combines several of the previous examples to show the potential use of the SRL16E primitive in a variety of modes.

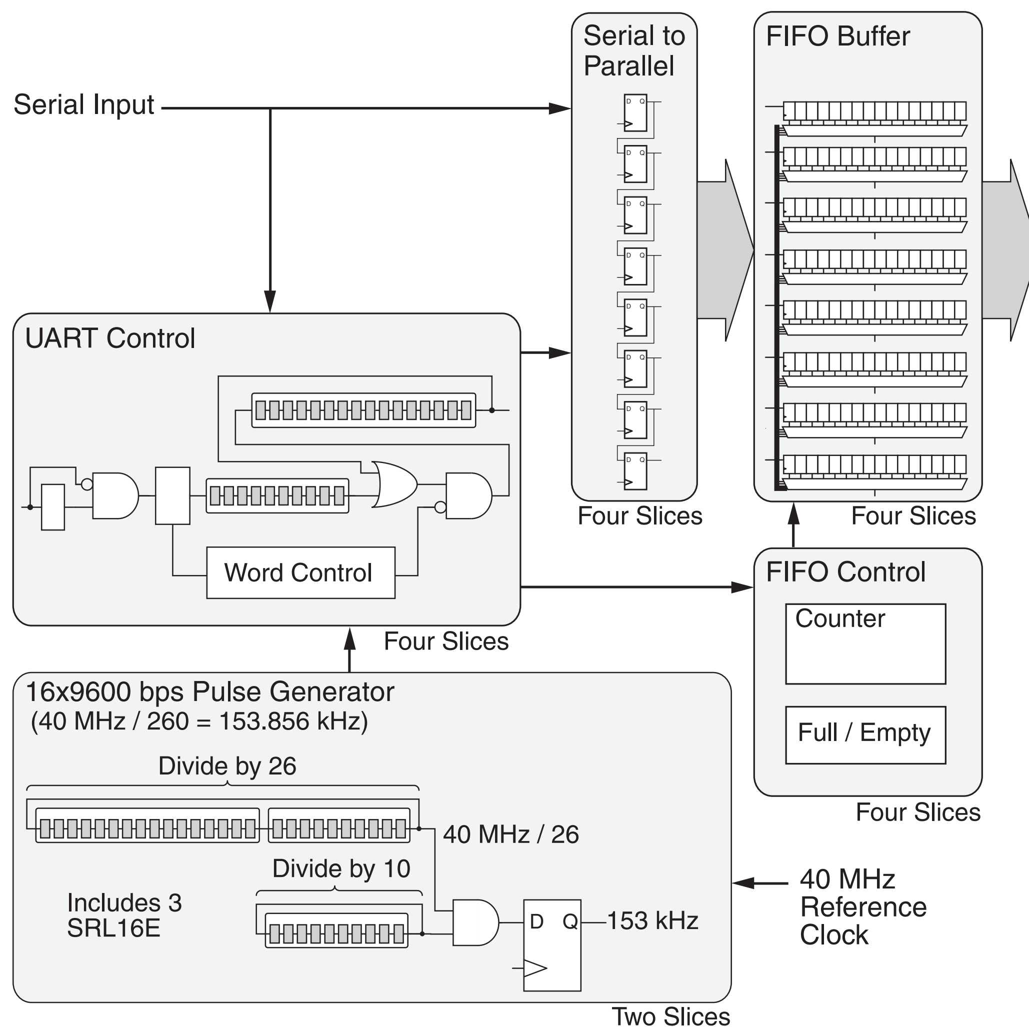

Figure 31: **Complete RS-232 Receiver**

Note that this example uses 18 slices to provide 207 flip-flops which yields a density of 69 gates per slice (based on just 6 gates per flip-flop). 13 Look-Up Tables are used in SRL16E mode, and 21 are used in standard logic mode.

Summary

The SRL16E is an alternative configuration of the Xilinx Look-Up Table, changing it from a 4-input Look-Up Table to a 16-bit shift register. The SRL16E configuration takes advantage of the inputs provided in order to use the Look-Up Table as RAM, which are supported in the SLICEM elements of the Spartan-3 Generation FPGAs. Although this configuration may often be used automatically, careful consideration in the design description can allow full access to this very efficient feature.

References

For more information, see UG331, Chapter 7, Using Look-Up Tables as Shift Registers (SRL16):
(http://www.xilinx.com/support/documentation/user_guides/ug331.pdf).

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
05/22/08	1.0	Initial Xilinx release. Originally published as a TechXclusive.

Notice of Disclaimer

The information disclosed to you hereunder (the "Information") is provided "AS-IS" with no warranty of any kind, express or implied. Xilinx does not assume any liability arising from your use of the Information. You are responsible for obtaining any rights you may require for your use of this Information. Xilinx reserves the right to make changes, at any time, to the Information without notice and at its sole discretion. Xilinx assumes no obligation to correct any errors contained in the Information or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE INFORMATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS.