WP273 (v1.0) February 1, 2008

Performance + Time = Memory (Cost Saving with 3-D Design)

By: Ken Chapman

"Performance + Time = Memory" is a strange formula, but when understood, it can often result in significantly lower cost implementations with Xilinx devices. While this is vital for DSP applications, I really like the way it can be applied to many designs. It is particularly useful for applications that use the range of Spartan™ devices, where cost savings are always welcome for high-volume applications. The same concepts can be applied to any FPGA or CPLD family.

In this white paper, I will investigate the formula and why it is so applicable to Xilinx devices. I will then describe a case study to consider, and potential solutions.

Parallel Design is Two-Dimensional

In most hardware designs, the Xilinx FPGA is treated as a two-dimensional fabric, as shown in Figure 1. Configurable Logic Blocks (CLBs) contain slices that are used to provide the logical functions, and blocks of RAM are used for buffers such as FIFOs. The tendency is for a design to become larger as more functionality is required. The larger design therefore uses larger devices. The clock speed can often be well below 100 MHz, and many of the functions are clock-enabled at even lower rates.

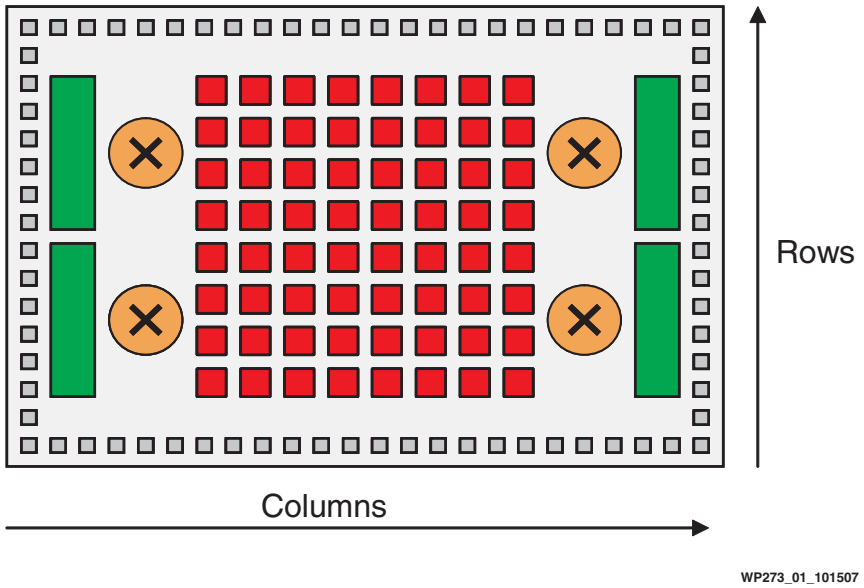

Figure 1: **Two-Dimensional FPGA Fabric**

As the cost of a design is proportional to the size of the device, parallel implementations, even if well optimized, will be relatively expensive. They cannot be avoided where maximum performance is required.

Applications, such as bus interfaces that need to have a predefined number of pins and clock rate, are also fundamentally constrained in the way that they can be implemented. However, when processing functions have a relatively long time period to be completed, such 2-D design is wasteful and unnecessarily expensive.

Parallel Hardware Designs Don't Obey the Formula

A parallel design provides logic for each and every function that must be implemented. This means that there is actually a zero requirement for memory, because a signal (wire) exists for every value to be calculated. In Figure 2, it can be seen that the value "A+B+C+D" is created; however, since it is immediately applied to the final adder, it does not need to be stored.

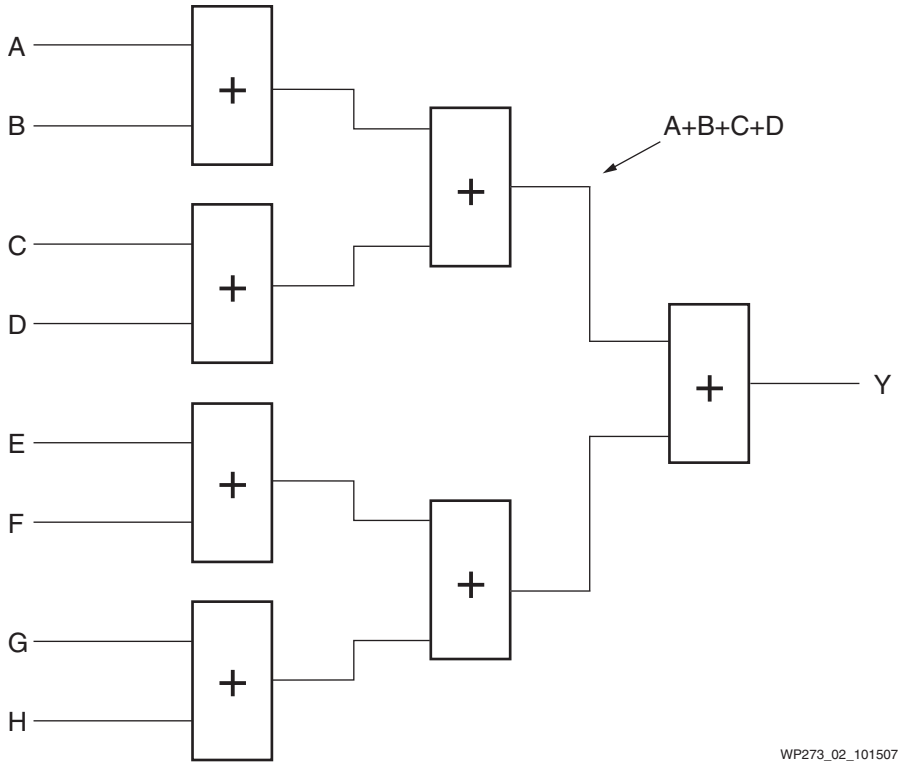

Figure 2: **Addition Tree Example of Parallel Design**

Of course, the parallel implementation offers the very highest performance. The adder tree above can easily exceed 100 MHz in a Spartan device, which is equivalent to over 700 million additions per second. However, such a structure cannot benefit from having more time to complete the required operation. The only benefit is that it will consume less power if it is clocked slower.

If there is 1 ms available to perform the addition tree, the design can be clocked at 1 KHz. It will work, but it wastes the Spartan FPGA silicon performance. Even worse, the more values that need to be added, the larger the circuit becomes; this increases the cost of the final product.

Processors Obey the Formula

Now, take a closer look at the familiar world of processors. A processor is a very good time-sharing engine. The ALU is directed to perform many different operations over many clock cycles to complete the desired process in the required time period, as shown in Figure 3. The higher the performance of the processor, the faster the ALU will be clocked and the more that ALU can be time-shared to achieve the algorithmic process.

For example, given that a particular process needs to be completed in a maximum time of 1 ms, the number of clock cycles available for the processor to exploit depends on the performance:

- Clock speed 1 MHz provides 1000 clock cycles per 1 ms.
- Clock speed 100 MHz provides 100,000 clock cycles per 1 ms.
- Clock speed 200 MHz provides 200,000 clock cycles per 1 ms.

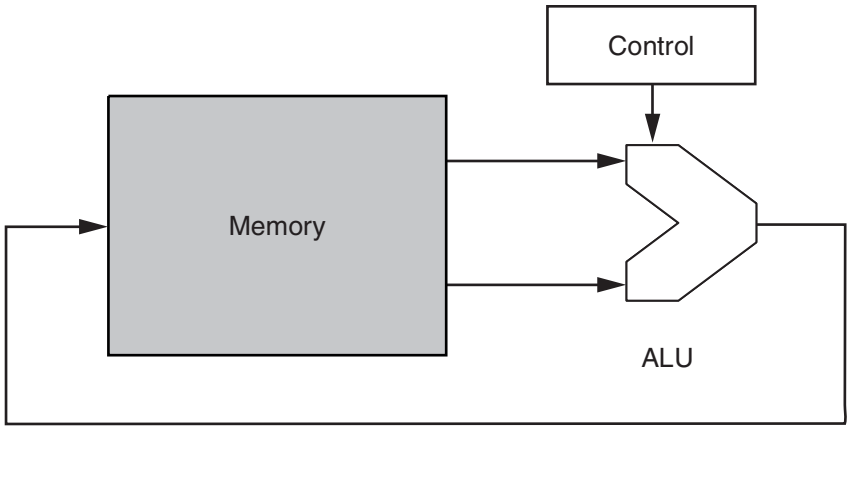

Figure 3: **Simplified Processor Example of Time-Sharing**

That is all very obvious, but less obvious is the direct link this has to memory. Suppose the available clock cycles are used by the ALU to perform the trivial task of summing data values. In a 1 ms time period, a 1 MHz clock rate means that the processor has the ability to sum 1,000 data values. It will have to get these values from somewhere, and that place will be memory. As the clock increases to 200 MHz, it can then use the same amount of logic to sum 200,000 data values; it now needs a memory to hold 200,000 words.

In a more realistic case, a processor tends to apply multiple instructions to each data set, so the memory requirement to store data is not so high; all the same, there is a very strong relationship.

Sequential Design is 3-Dimensional

Making the decision to operate the logic functions at a higher rate than the processing rate allows operations to be achieved sequentially. As with a processor, logic is time-shared over multiple clock cycles. Since "Performance + Time = Memory," we also need to use memory to hold all the values not being used on a given clock cycle and often partial/temporary results created during the processing.

The FPGA can now be considered to be a three-dimensional volume to be filled, as shown in Figure 4.

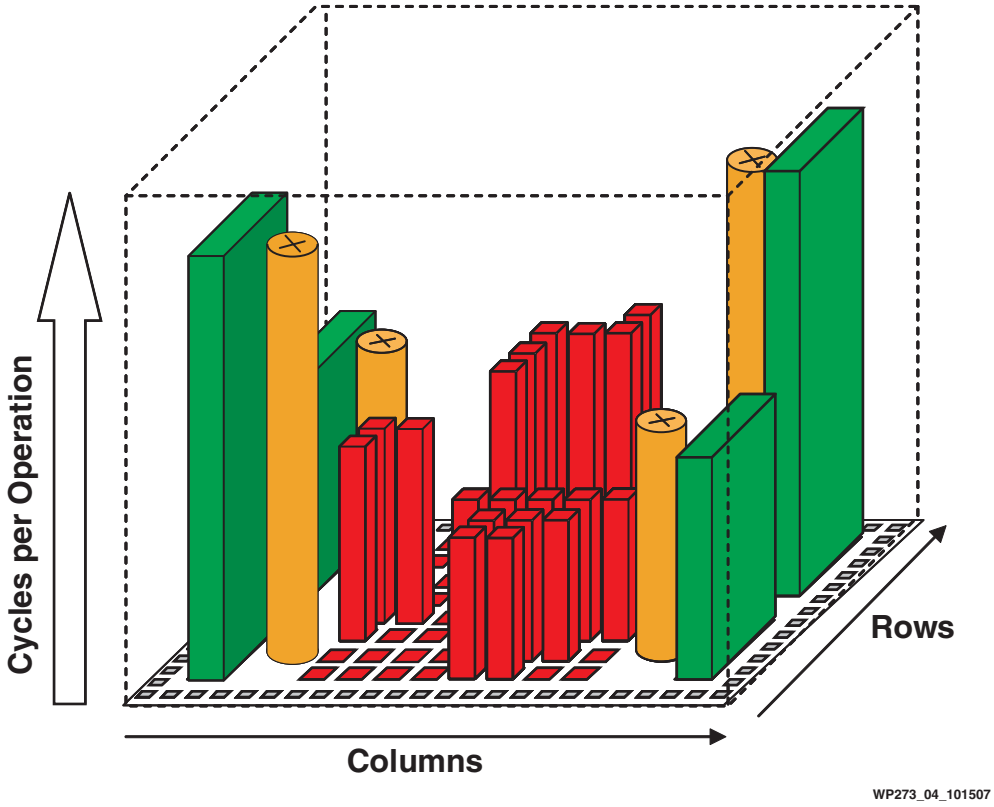

Figure 4: **Three-Dimensional FPGA Volume**

The advantage is that you only pay for the 2-D fabric being occupied. The only limits to "building" upwards are the maximum clock rate of the device and the amount of RAM available in a given block. As well as the dedicated blocks of RAM, each CLB can be used to provide distributed RAM (or my favorite SRL16E). This allows the correct amount of memory to be allocated in each position and prevents memory access bottlenecks from forming in the design.

Approach to Design

When any function is implemented, two questions should be asked:

1. How much time is available to complete the process?
2. Given the performance of the selected Xilinx device, what clock rate will be used?

The answer to the first question comes from the design specification. The way in which you partition the design into functions can have quite an impact, so do consider some alternatives. As to the performance of Xilinx FPGA devices, this has more to do with "design comfort" than the actual peak performance of the devices.

Personally, I like to see devices clocked above 75 MHz, and I find this relatively easy to achieve. The higher the clock rate, the more challenging the design; however, anything less than a 50 MHz clock rate is really getting very slow and wasteful of the performance offered from any of the devices. Remember that the embedded DLL (Delay Locked Loop) and DCM (Digital Clock Manager) blocks can be used to create internal clocks of a higher rate than available on the PCB.

The answers to the above questions will let you know if there is any potential for time-sharing of logic resources. This leads to a third and final question:

3. How can **memory** resources of the device be utilized to reduce the size of implementation?

Now the engineering starts. It does take some practice, so here is a design specification for you to consider.

Design Challenge

The challenge is to design a small box to be used in factories that process items such as fruit and vegetables. The card is used to collect data from light sensors located on the various conveyor belts along which the fruit and vegetables pass as they are sorted for type, quality, and size, as shown in Figure 5.

The initial design concept is to employ a microcontroller (or similar small processor) to collate the information and communicate it via serial (RS232) links to a PC in the factory control room. An FPGA is being considered to interface the processor to the sensors. The product is required in high volume (50,000 to 100,000 units), so a Xilinx Spartan-3A FPGA is the target for a cost-effective solution.

The card supports 64 sensors. A logic "1" signal is generated when the light beam is broken by a passing object. The maximum speed of the conveyor belt is 1 meter per second. The minimum width of a single item is 3 cm, and there is minimum of 10 cm space between items on the belts.

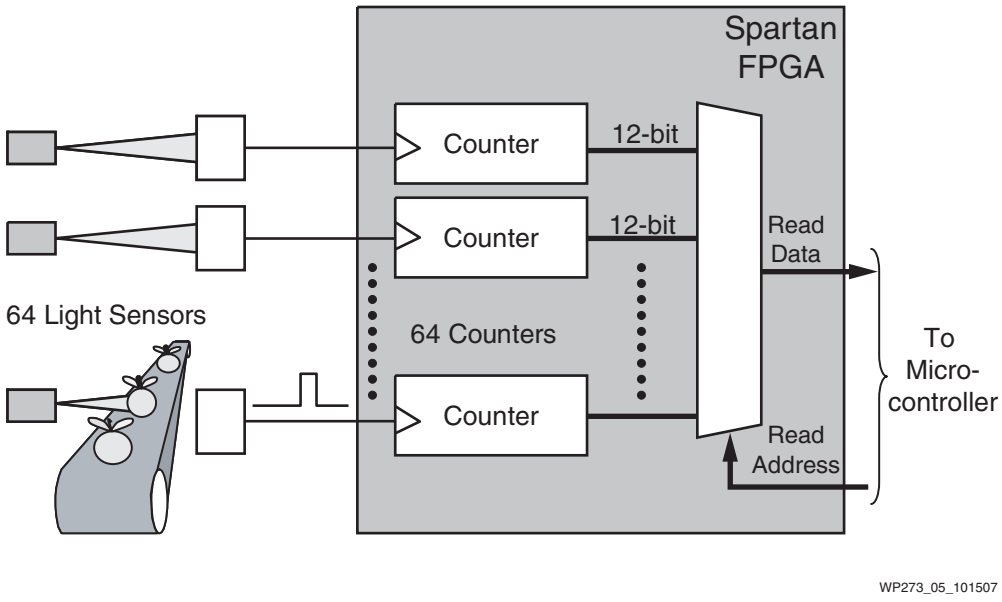

Figure 5: **Factory Control Challenge**

Each pulse is used to increment a separate counter, which can support a maximum value of 4095 (12-bits). A simple interface to the microcontroller is then able to read the value of any of the 64 counters in the card by supplying a 6-bit address.

Initial Observations

By taking a very direct approach to the design, we could simply identify the need to implement 64 counters of 12-bits followed by a 64:1 data multiplexer. In fact, this is a direct representation of the block diagram in Figure 5.

There is a need to apply some fundamentally good engineering, because we certainly wouldn't want to have 64 independent clocks in a design. This would lead to very poor utilization of the device and have a high probability of unreliable operation. The signal inputs really should be synchronized to a single internal clock, and then clock enables should be used with the counters.

First Estimate

An estimate of the device resources used can easily be made with a basic understanding of the device architecture.

- **Counters** - Since each slice can implement a 2-bit counter, six slices are required to implement a 12-bit counter. Therefore, a total of 384 slices are required for all 64 counters.
- **Multiplexer** - Each slice contains two look-up tables and a dedicated multiplexer (MUXF5), enabling a 4:1 multiplexer to be implemented. However, each pair of slices within a CLB share an additional dedicated multiplexer (MUXF6), enabling a complete 8:1 multiplexer to be implemented in two slices. Dedicated MUXF7 and MUXF8 multiplexers allow a 32:1 mux in only eight slices. Two of these 32:1 multiplexers and another 2:1 mux in 0.5 slices are required to construct a 64:1 multiplexer, which must be replicated 12 times to support the data width of the counters. The total size of the multiplexer is then 16.5 × 12 = 198 slices.
- **Synchronizing Logic** - At this stage, we have not designed the logic to capture the input signals and synchronize them to the internal clock. For now, we will allow a slice per input (2 flip-flops and some gates) for a total of 64 slices.

Based on these major building blocks of the design, our estimate is for 646 slices. This indicates that the smallest Spartan-3 Generation FPGA can be used. An XC3S50A device is suitable with its 704 slices, providing a surplus of 58 slices to complete the processor interface.

Performance + Time = Memory

So now, take a while to consider the formula. Ask the right questions, and see if it can help lower the costs on this design. Can you see a way to make a 3-D design to lower the cost? Once you have come up with your own solution, continue reading.

I will show you my ideas, and you can see if they match up with yours.

Answering the Challenge

Well, have you been able to see a way to use the "Performance + Time = Memory" formula to lower the cost of the challenge design?

The following sections describe one way of implementing the design. As soon as you work out that such a 3-D design is possible, there really are many ways to implement it. So, don't worry if you have something a bit different. As long as you have been able to significantly lower the cost, you are well on your way to improving the profit margins on your own designs in future. Remember, the target to beat is 646 slices in a Spartan-3 XC3S50A device, which was the result of a full parallel 2-D design.

We must begin our 3-D design process by asking the right questions that relate to the "Performance + Time = Memory" formula.

How Much Time Is Available?

Taking the minimum fruit size and minimum spacing between fruit passing on a belt at the maximum speed of 1 meter per second, we derive the timing of the fastest pulses from a light sensor, as shown in Figure 6.

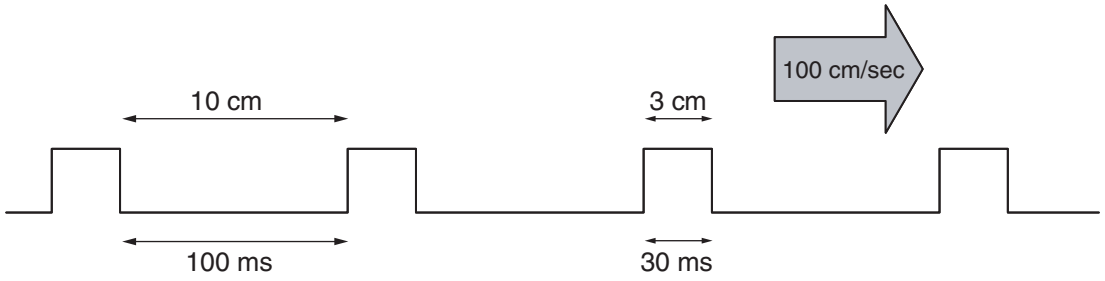

Figure 6: **Factory Control Timing**

We soon discover that the pulses are of a long duration and that the pulse rate is very low. In fact, the maximum pulse rate is less than 8 Hz, which is very slow indeed. However, we must consider that there are 64 sensors to be monitored, and we could be unlucky enough to have them all triggered at the same time. So, all 64 sensors must be serviced in a maximum of 30 ms, and the aggregate data rate is more like 500 Hz.

What Performance Is Available?

We know that the Spartan-3A family is the target architecture. These devices are capable of operation above 100 MHz, so device performance should not limit us at all in this case. While I want to work the silicon hard to get the most out of it, there is no point overdoing it and burning power unnecessarily. In this case, it is better to work out the minimum clock rate required to process all 64 channels, and then tie this in with a suitable clock source on the PCB.

Looking at the timing waveform, the pulse width caused by the smallest fruit breaking the light beam is the most demanding. I must guarantee that I observe each sensor at least once every 30 ms. If the 64 sensors are observed and processed sequentially,

rather than in parallel, then 30 ms/64 is the maximum time that can be allocated to each sensor. This means that the minimum processing rate is 2133 Hz. Obviously, this is still desperately slow, but only emphasizes that "Performance + Time = Memory" must be a valid formula to be applied in this case.

Replacing Counters with Memory

We have "Time" and we have ample "Performance," so now it is a case of working out how to make the whole thing a sequential 3-D design. How can memory resources of the device be utilized to reduce size of implementation?

Because memory is used to hold data values, we must identify where the "data" is in the system. These may be complete values or partial values, so have a good look through the block diagram and identify them. In this system, they are fairly obvious in that the counters each hold a value. In the parallel implementation, they are distributed across the 384 slices forming the 64 counters, but we want to consolidate them into a single memory.

You can choose between distributed (CLB) memory and dedicated (block) memory, and we could really use either to form storage for 64 values of 12-bits. However, as the dedicated block RAM isn't required for anything else, I have taken that option. Configured as 1024 words of 18-bits, a single block provides more than adequate storage, as shown in Figure 7.

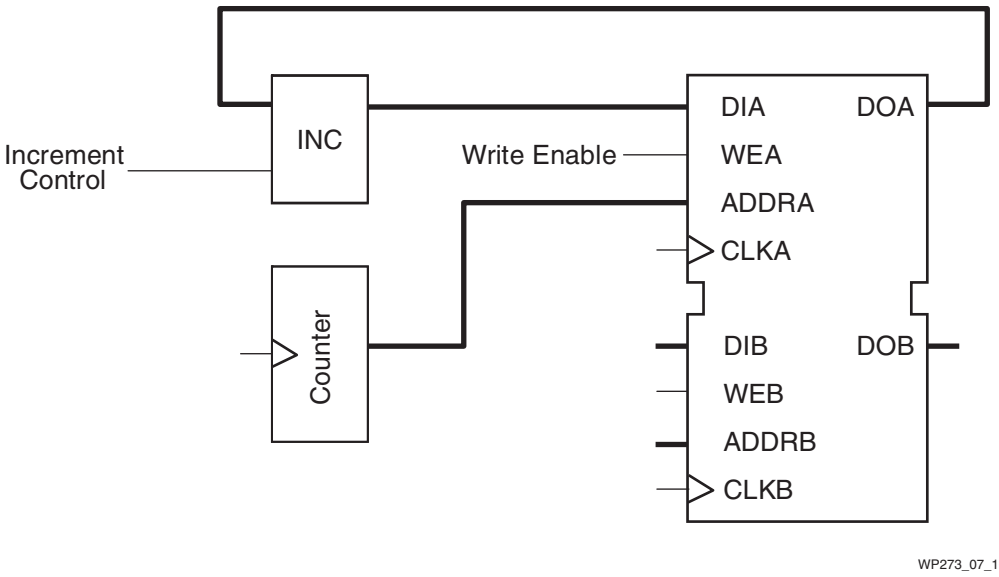

Figure 7: **Factory Control Storage in Block RAM**

The counter functionality is then replaced by a single increment function. A "count value" is read from the RAM, passed through the increment block, and then written back into the RAM at the same location. This is best organized as a two-cycle process, but this is not an issue because of the "Performance + Time" that is available.

Although we could selectively access the count values to be incremented as a corresponding light beam is broken, it is much easier to scan sequentially through all 64 count values and increment only those which must be increased before the value is written back into the RAM. This reduces the address generation to a simple 6-bit counter.

At this stage, we have replaced 384 slices with one block RAM and just nine slices of logic (six for the increment and three for the address counter). This is a huge saving. Now, however, we must find ways to connect the inputs and outputs to this 3-D processing engine.

Free Data Multiplexer

The parallel data multiplexer is simply not required in this design. Because the count values are now held in one consolidated memory, 198 slices are instantly saved. The dual-port nature of the block RAM really makes it very easy to connect the external processor, as shown in Figure 8.

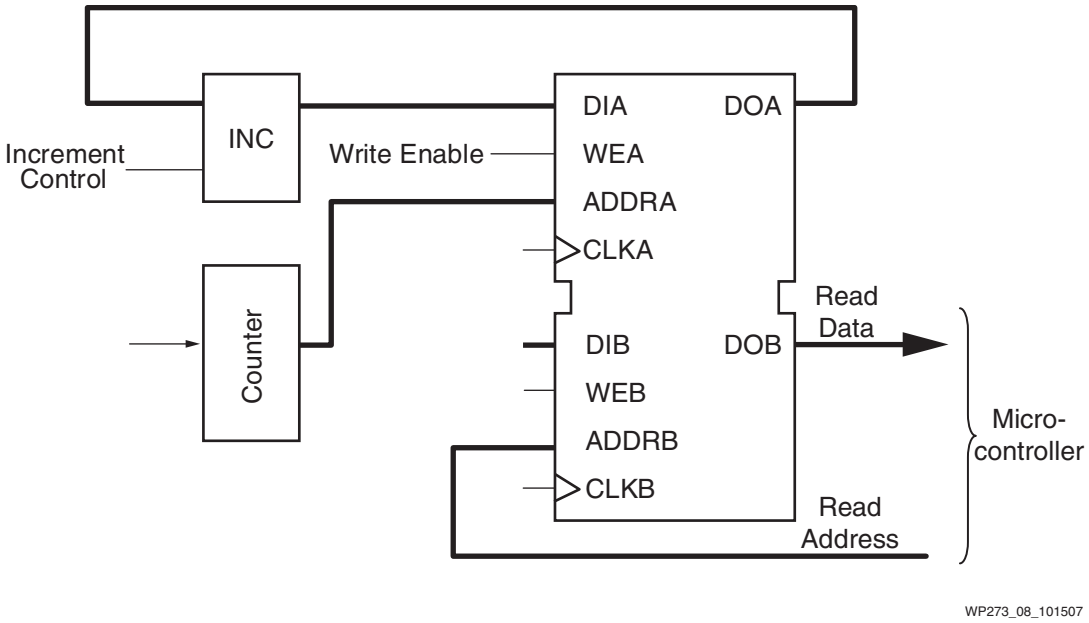

Figure 8: **Factory Control Connection to Processor**

The memory also offers the opportunity for the processor to have a write mode to reset count values or set test values. As with the parallel implementation, there is a risk that the processor will try to read a count value that is in the process of being modified. However, it would be very easy to allocate time for the increment process and time for the processor to read values. Although a clock rate of a few kHz is adequate for the processing, a clock of 2 MHz (or similar clock rate associated with the microcontroller) would achieve a count value update scan in 64 µs leaving nearly all of the 30 ms processing period available for the microcontroller to read or write count values.

Connecting the Sensors

At some point in all these 3-D designs, the parallel world must be connected to the sequential processing engine. This does not have to be difficult, and often the simple method is adequate.

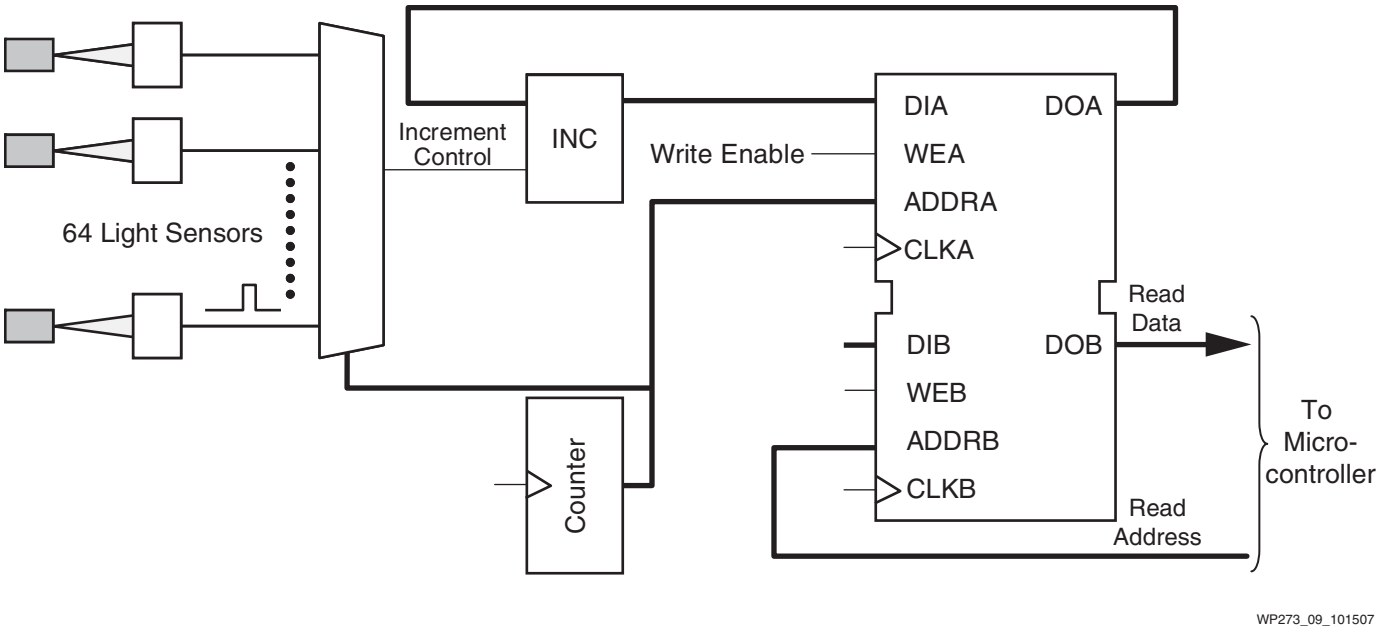

Figure 9: **Factory Control Connection to Sensors**

The counter used to access each count value from the RAM can be used to select the associated sensor via a 64:1 multiplexer, shown in Figure 9. Although this requires multiplexer logic, it is only for one bit and therefore only requires 16.5 slices.

Each sensor still requires its own logic. This is partly to synchronize the input signals, but is also required to ensure that each "beam broken" pulse is only used to increment a count value once. For this reason, the one slice per sensor allotment is unlikely to be reduced.

When you see that the logic size is increasing because the function is becoming more parallel, it is worth looking to see if anything else can be time-shared and moved into memory. In this case, we can indeed improve things, as shown in Figure 10.

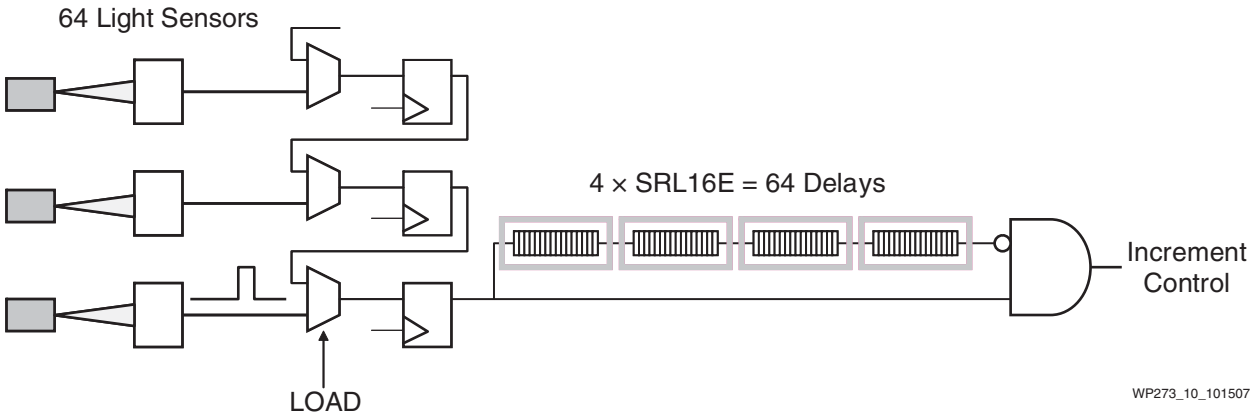

Figure 10: **Sensor Multiplexer Replaced by SRL16E**

The multiplexer is replaced by a 64-bit parallel to serial converter (32 slices), which converts the parallel domain into a serial sequential process. In order to detect only the start of a new pulse, a memory is used to remember the last state of each of the 64 sensors. Since the operation is so predictable, we can use the SRL16E memory mode, which requires just two slices.

Reduction in Cost

So was it worth it? I think Figure 11 and Figure 12 speak for themselves.

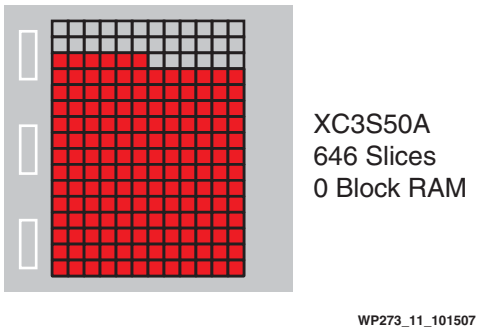

Figure 11: **Original 2-D Design**

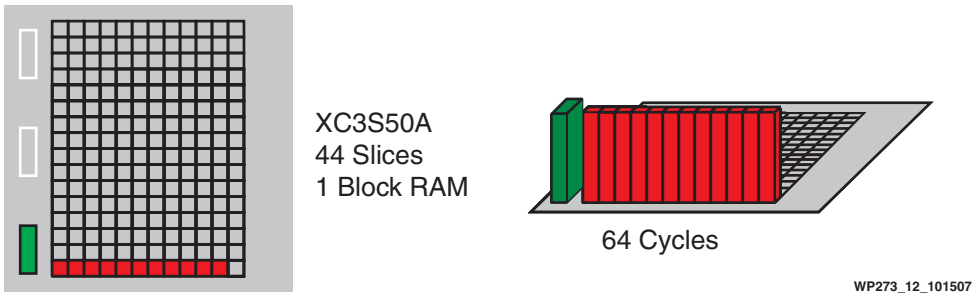

Figure 12: **Sequential 3-D Design**

To reduce the function from 646 slices to just 44 slices is a dramatic saving: 15 times smaller.

However, this is not just specific to this particular design. For example, so much of the 3G-wireless processing involves "chip rates" of 1.2288 MHz and 3.84 MHz. This provides the "Time" to allow the "Performance" and "Memory" of FPGA devices to process at least 32 channels sequentially in just the same way as this simple fruit counter has shown.

Final Considerations

The XC3S50A is available in a TQ144 package. However, this only provides 108 user I/Os, and the design has high I/O demands. Having used 64 for sensor inputs and applied a clock, only 43 are left for the microcontroller interface. Given an 8-bit data bus, it is possible to connect to the microcontroller; however, it does illustrate how I/Os can limit a design once these highly efficient techniques are employed.

Of course, it would be a pity for 94% of the XC3S50A to be so completely wasted. It would be nice to embed the microcontroller and the UART in the same device. Along with the obvious product cost reductions resulting from removing the microcontroller and simplifying the PCB, this would lower the demands on I/Os. It is possible to implement the UART in 40 slices, but surely a microcontroller is beyond a Spartan device.

If only it were possible to implement a microcontroller in a portion of the 660 slices and two block RAMs remaining in the XC3S50A. If you are interested, take a look at http://www.xilinx.com/picoblaze to download and use a microcontroller that fits in just 96 slices.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
2/1/08	1.0	Initial Xilinx release. Some content taken from previous web postings as a TechXclusive.

Notice of Disclaimer