

Alpha Blending Two Data Streams Using a DSP48 DDR Technique

Achieve full throughput of the DSP48 slice with a double-data-rate technique.

by Reed Tidwell
Sr. Staff Applications Engineer
Xilinx, Inc.
reed.tidwell@xilinx.com

The XtremeDSP™ system feature, embodied as the DSP48 slice primitive in the Xilinx® Virtex-4™ architecture, is a high-performance computing element operating at an industry-leading 500 MHz. The design of the Virtex-4 infrastructure supports this rate, with Xesium clock technology, Smart RAM, and LUTs configured as shift registers.

Many applications, however, do not have data rates of 500 MHz. So how can you harness the full computing performance of the DSP48 slice with data streams of lower rates?

The answer is to use a double-data-rate (DDR) technique through the DSP48 slice. The DSP48 slice, operating at 500 MHz, can multiplex between two data streams, each operating at 250 MHz.

One application of this technique is alpha blending of video data. Alpha blending refers to the combination of two streams of video data according to a weighting factor, called alpha. In this article, we'll explain the techniques and design considerations for applying DDR to two data streams through a single DSP48 slice.

All Virtex-4 devices have DSP48 slices, although the SX family contains the largest number (an industry-high 512) and the highest concentration of DSP48 slices to logic elements, making it ideal for math-intensive applications ...

Virtex-4 DSP48

The DSP system elements of Virtex-4 FPGAs are dedicated, diffused silicon with dedicated, high-speed routing. Each is configurable as an 18 x 18-bit multiplier; a multiplier followed by a 48-bit accumulator (MACC); or a multiplier followed by an adder/subtractor. Built-in pipeline stages provide enhanced performance for 500 MHz throughput – 35% higher than for competing technologies.

All Virtex-4 devices have DSP48 slices, although the SX family contains the largest number (an industry-high 512) and the highest concentration of DSP48 slices to logic elements, making it ideal for math-intensive applications such as image processing.

A triple-oxide 90 nm process makes the DSP48 slice very power-efficient.

Architectural features, including built-in pipeline registers, accumulator, and cascade logic nearly eliminate the use of general-purpose routing and logic resources for DSP functions, and further reduce power. This slashes DSP power consumption to a fraction when compared to Virtex-II Pro™ devices.

DDR with Two Data Streams

DDR, in this context, refers to multiplexing two input data streams into one stream at twice the rate, interleaving (in time) the data from each stream (Figure 1). Figure 1 also shows the reverse operation, creating two parallel resultant streams after processing.

You can drive the DSP48 slice inputs at the fast 500 MHz clock rate from CLB

flip-flops; CLB LUTs configured as shift registers (SRL16); or directly from block RAM. Block RAM, configured as a FIFO using the built-in FIFO support, also supports the 500 MHz clock rate.

Design Considerations

Dealing with data at 500 MHz requires great care; you should observe strict pipelining with registers on the outputs of each math or logic stage. The DSP48 slice provides optional pipeline registers on the input ports, on the multiplier output, and on the output port from the adder/subtractor/accumulator. Block RAM also has an optional output register for efficient pipelining when interfaced to the DSP48 slice.

Where you are using CLBs, place only minimal levels of logic between registers to provide maximum speed. For DDR operation, only a 2:1 mux (a single LUT level) is required between pipeline stages. Whether you are interfacing to the DSP48 slice with memory or CLBs, placing connected 500 MHz elements in close proximity minimizes connection lengths in the general routing matrix.

DDR requires the DSP48 slice to operate at double the frequency of the input data streams. You can use a DCM to provide a phase-aligned double-frequency clock using the CLK 2X output.

Another aspect of inserting DDR data through a section of pipeline is ensuring that data passes cleanly between clock domains. This may require adding extra registers clocked with the double-frequency clock at the output of the double-pumped section, to synchronize the data with the original clock. The rule of thumb is that in order to insert a double-pumped section cleanly into a single-pumped pipeline, there must be an even number of register delays in the double-pumped section.

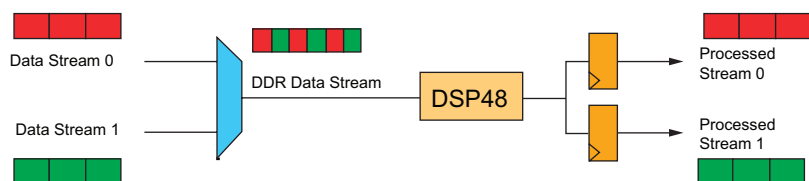


Figure 1 – DSP48 DDR

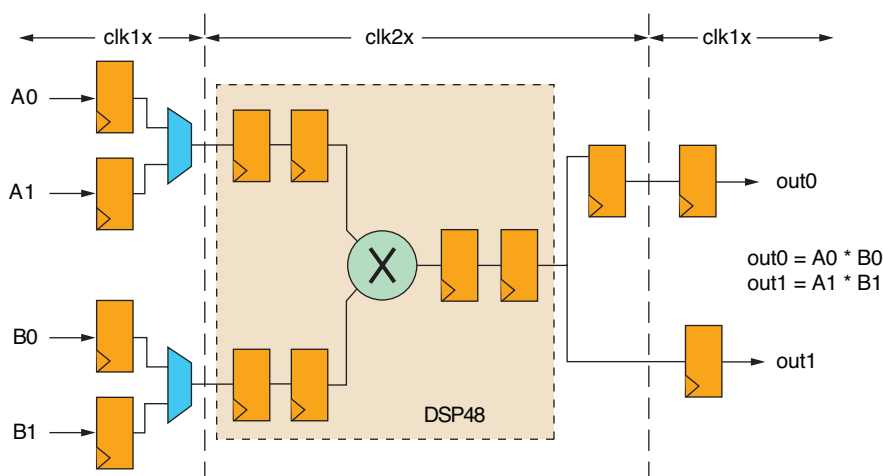


Figure 2 – Two-stream multiply through DSP48 slice

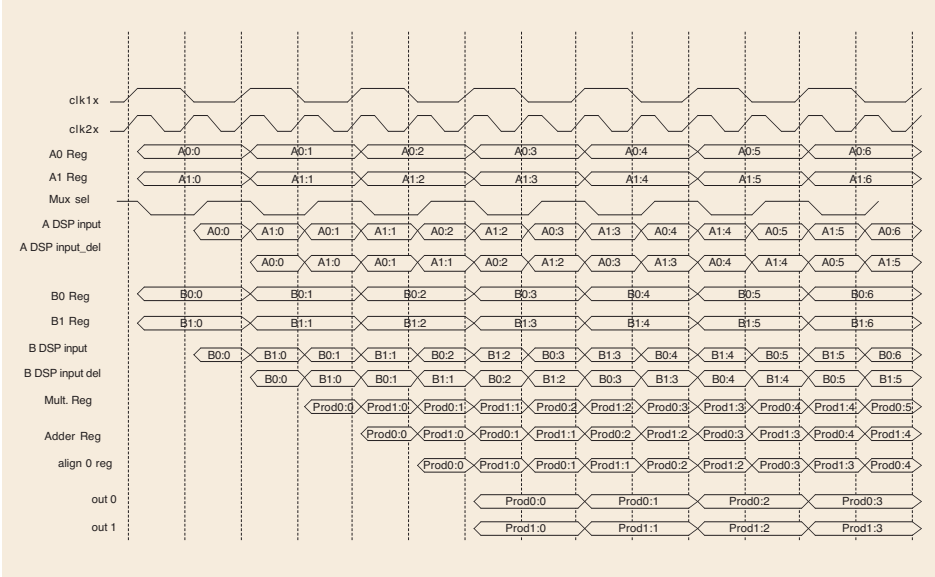


Figure 3 – Timing of two-stream multiply

Implementation

Several configuration options exist for implementing DDR functionality. Figure 2 shows a straightforward implementation.

In Figure 2, stream 0 consists of A0 and B0 inputs. We multiply them together and output as out0. Likewise, stream 1 consists of inputs A1 and B1 multiplied together and output as out1. There are two clock domains: the clk1x domain, at the nominal data stream frequency, and the clk2x domain, at twice the nominal frequency.

Figure 2 shows two registers after the multiplier. The second is the accumulation register, even though we do not use accumulation in this configuration. The register, however, is still required to achieve the full, pipelined performance. We use two sets of registers on the inputs of the DSP to make the total delay through the DSP48 slice an even number (four) for easier alignment of the output data with clk1x. These registers are “free” because they are built into the DSP48 slice, and using them reduces the need for alignment registers external to the DSP48 slice. The extra pipeline register on out0 compensates for taking stream 0 into the DSP one clk2x cycle before stream 1. As seen from the timing diagram in Figure 3, this is required to realign the stream 0 data back into the clk1x domain.

Note that the input mux select, mux_sel, is essentially the inverse of clk1x. It is important, however, to generate this signal from a register based on clk2x (rather than deriving it from clk1x) to avoid hold-time violations on the receiving registers.

At the transitions between clock domains, the data have only one clk2x period to set up. This is the reason to have no

logical operations between registers in the two domains. The placement of the first registers in the clk1x domain is more critical than other registers in the same domain.

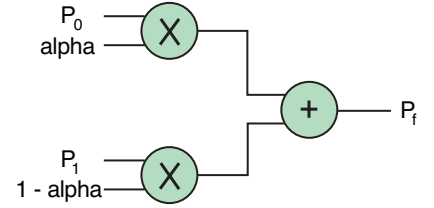


Figure 4 – Alpha blend formula in graphical terms

Alpha Blending

Alpha blending of video streams is a method of blending two images into a single combined image, such as fading between two images, overlaying anti-aliased or semi-transparent graphics over an image, or making a transition band between two images on a split-screen or wipe. Alpha is a weighting factor defining the percentage of each image in the combined output picture. For two input pixels

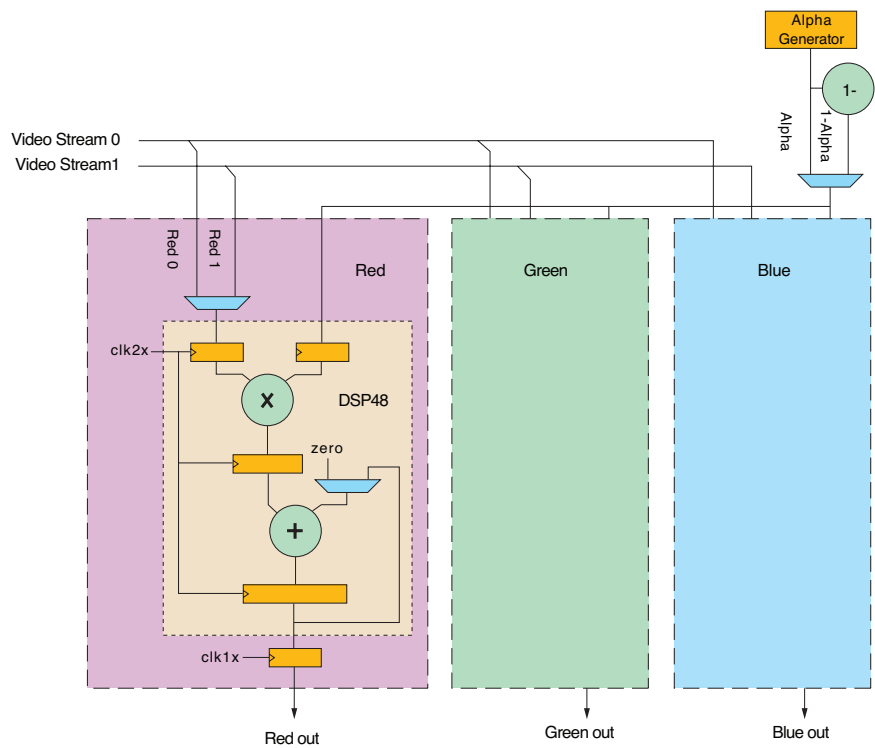


Figure 5 – Alpha blend on three-component video

You can efficiently use the high-performance of Virtex-4 devices with DSP48 slices by processing multiple data streams in a time-multiplexed fashion.

(P_0 , P_1 , and a blend factor, α , where $0 \leq \alpha \leq 1.0$), the output pixel P_f will be:

$$P_f = \alpha P_0 + (1-\alpha)P_1 \text{ (see Figure 4)}$$

This operation is performed separately for each component: red, green, and blue.

A pixel rate of 250 MHz or less is sufficient for all standard and high-definition video rates, and common Video Electronics Standards Association (VESA)

standards as high as 1600 x 1200 at 85 Hz. Therefore, one DSP48 slice can perform the multiply and add on one component, and a set of three slices can alpha blend the three components from each of two video streams, as shown in Figure 5. The operations must be performed identically and in parallel on each of the three components.

There are several ways to implement alpha blending depending on the nature

of the video streams and how alpha is generated. Figure 6 shows a basic implementation with two video streams alternating as one multiplier input. The other multiplier input alternates between alpha and 1-alpha.

The operating mode of the adder alternates between add zero (pass through) mode and add output (accumulate) mode. The DSP48 slice output register contains the result of the $\text{Video0} * \alpha$ multiply during one clock cycle, and the final result ($\text{Video1} * (1 - \alpha) + \text{Video0} * \alpha$) on the alternate clock. Figure 7 shows the timing for this configuration.

The align registers on the inputs of the DSP are used to make the total delay through the DSP48 slice an even number (four), as explained in the previous example. The final output register for blend loads new data to every other DSP clock to register the blend results at the original pixel rate.

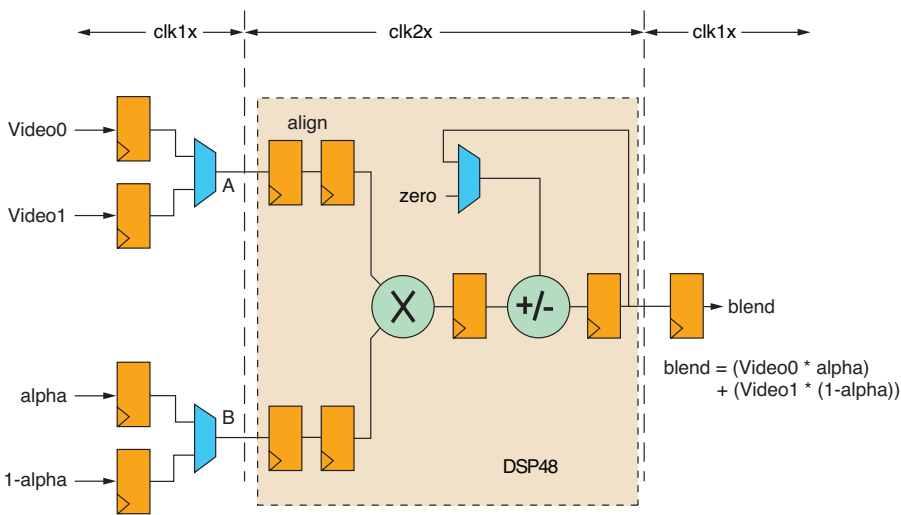


Figure 6 – Alpha blend implementation (one component)

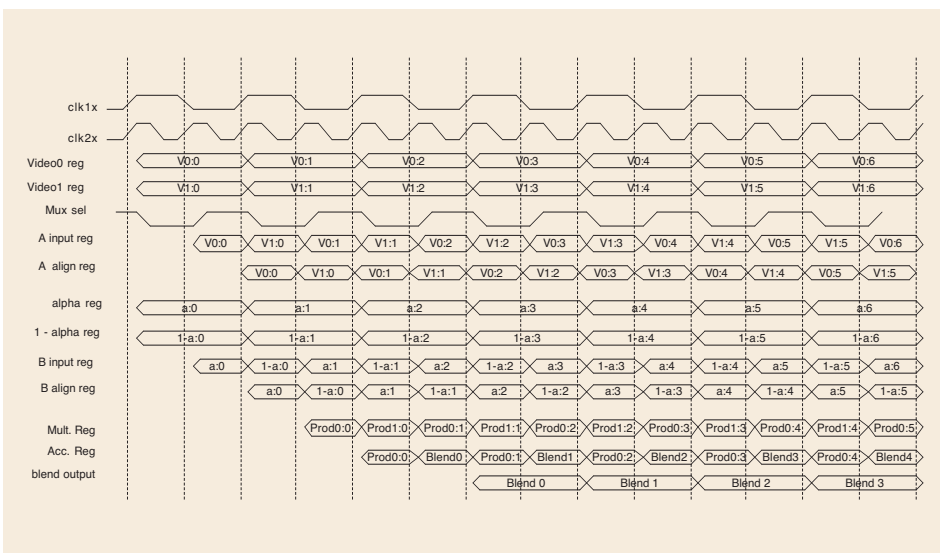


Figure 7 – Alpha blend timing

Conclusion

You can efficiently use the high-performance of Virtex-4 devices with DSP48 slices by processing multiple data streams in a time-multiplexed fashion. With careful design, a single DSP48 can perform multiply operations on two independent data streams, operating at 250 MHz each.

Alpha blending of video streams, as outlined in this article, is one example of processing two data streams through a single DSP48 slice. This capability complements the DSP features of Virtex-4 FPGAs – including built-in pipelining and cascading, integrated 48-bit accumulator, and an abundance of DSP48 slices in the SX family – to make Virtex-4 devices the ideal DSP platform.

For details about the DSP48 slice, refer to the “Virtex-4 FPGA Handbook,” Chapter 10, or the “XtremeDSP Design Considerations User Guide” at www.xilinx.com/bvdocs/userguides/ug073.pdf.