

The Need for Dynamic Phase Alignment in High-speed FPGAs



With bandwidth demands increasing at logarithmic scales, high-data-rate devices are not an option — they are a necessity. Timing is a significant issue in high-speed FPGAs that needs to be managed in order to sustain multi-gigabit data rates and device reliability while containing costs and managing manufacturability.

With the explosion of data, voice and video traffic, FPGAs are ideal devices for data transmission applications where speeds in the Gb/s range and beyond are required. Well-suited for the job, FPGAs provide the flexibility to handle rapidly evolving standards in this design space.

Programmable logic solutions allow developers to rapidly bring products to market by avoiding long development times, high NRE costs and inventory risks associated with custom solutions. FPGAs with embedded silicon implementations of DPA advance the capabilities of programmable logic and complement, or, in some cases, provide an alternative to, high-speed CDR transceiver technology for multi-gigabit signaling. This article discusses the advantages of DPA technology and the benefits of incorporating it into FPGA products.

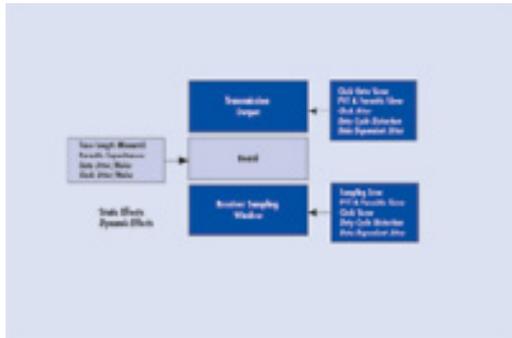
Timing is Everything

When building high performance products, system developers face challenges in maintaining the precise timing and signal integrity required to reliably sustain multi-gigabit data rates. Differential signaling standards, such as LVDS, help by providing common mode rejection, which greatly reduces the effects of electrical noise. CDR transceiver implementations combine the clock and data into a single signal, thus ensuring simultaneous arrival at their destination. However, a number of chip-to-chip interface standards are source-synchronous, which require a clock from the transmission source which is separate from the data. For these source-synchronous interfaces, keeping the clock and data signals in phase can be difficult when confronting board- and device-level effects such as skew, jitter and noise. For FPGAs

The Need for Dynamic Phase Alignment in High-speed FPGAs

Published on Wireless Design & Development (<http://www.wirelessdesignmag.com>)

to maintain their value in high-data-bandwidth applications, they require an integrated solution to address this design situation.



[1]

The increasing data rates required by many current designs result in shrinking margins for setup and hold times, reducing the available timing budget for factors such as skew, jitter, and noise. Skew is the difference in arrival time of bits transmitted at the same time is generated by a number of sources, both on the chip and on the board. These sources include variation in board trace lengths, use of connectors, differences in the propagation delays of the device I/O pins, variations in high-to-low or low-to-high transition times, parasitic circuits and the inability of clock management circuitry to instantaneously react to input changes. Jitter is the deviation from the ideal timing of an event is also derived from several sources in a digital system, both deterministic and random, including duty-cycle distortion, imperfect PLL output, ISI and EMI, such as crosstalk.

On boards with imperfect power supplies and reference planes, electrical noise may also be a factor. On the device level, each of these factors (skew, jitter and noise) is also influenced by variations in process, voltage and temperature, which further add to their impact. Figure 1 illustrates some of the sources of skew, jitter and noise in a digital system, indicating which are static and which can be dynamic.

Static Phase-alignment Methods

Several methods exist for reducing or eliminating the effects of skew, jitter and noise via static phase alignment. First, systems designers often attempt to match the length of clock and data traces during board layout. This method can reduce clock-to-data skew by making the distances traveled by the two signals equivalent, but it is labor-intensive, time-consuming and difficult, especially with today's high pin-count devices. This method also relies on a highly accurate analog simulation of the board. The simulation must account for long-term effects such as temperature changes. Even under the best circumstances, simulation methods can only offer an

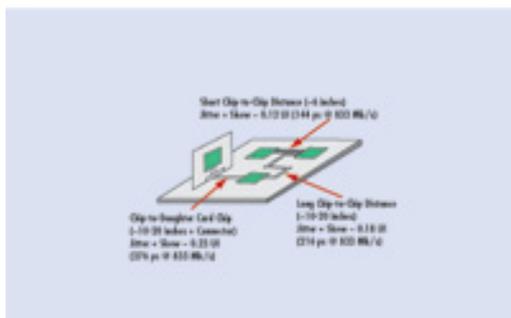
The Need for Dynamic Phase Alignment in High-speed FPGAs

Published on Wireless Design & Development (<http://www.wirelessdesignmag.com>)

estimation of the actual board conditions.

On the device level, vendors often provide integrated circuitry to shift the phase of the clock a fixed amount (such as 90°) on either the transmit or receive side, to align it with the data. On the transmit side, this method relies on matched trace lengths to be effective. The fixed amount of phase shifting results in little flexibility for the board designer, so some FPGA manufacturers have responded by allowing a variable amount of phase shift that can be configured at device startup. Although this programmability has increased the usefulness of phase shifting the clock on the receive side, it is limited by the resolution of the capture clock and does not provide for the long-term variations in skew that may result from process, voltage and temperature changes.

Although adequate for some designs, static phase-alignment methods are insufficient for higher-speed designs that require data rates in the range of 700 Mb/s and higher. In these applications, the increased clock frequencies have reduced the data window (or “eye”) for setup times to the point where it is difficult to maintain reliable operation. Consider a theoretical design that requires a data rate of 833 Mb/s, which therefore has a UI of 1.2 ns.



[2]

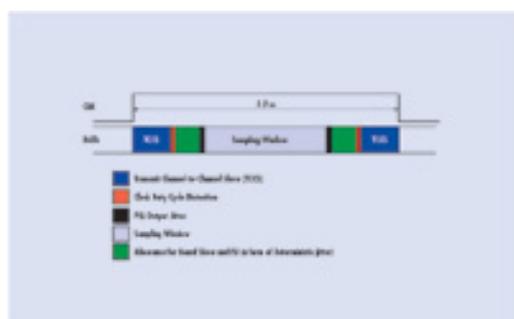
Figure 2 shows a component diagram of this system, including multiple chip-to-chip distances, as well as a daughter card that communicates with the main board through a connector. As indicated in Figure 2, the separate chip-to-chip paths cause various amounts of skew and jitter, ranging from around 0.12 to 0.32 UI, or about 144 to 384 ps. Given a UI of 1.2 ns for this design, an analysis of the associated timing budget indicates little or no tolerance for the typical amount of board skew and jitter that might be present in the system.

Figure 3 shows a diagram of the timing budget of this 833 Mb/s system and quantifies the amount of skew and jitter that can result from the components in the

The Need for Dynamic Phase Alignment in High-speed FPGAs

Published on Wireless Design & Development (<http://www.wirelessdesignmag.com>)

path of a given signal. The budget is composed of several elements. First, the transmit channel-to-channel skew is the amount of skew between different channels in the same clock domain, and includes effects from PVT, parasitic circuits (called parasitic skew), data jitter and variations in the periodic uniformity of the data waveform from the ideal case (called data duty-cycle distortion). Another component of the budget is variations in the clock signal from clock duty-cycle distortion and PLL output jitter.



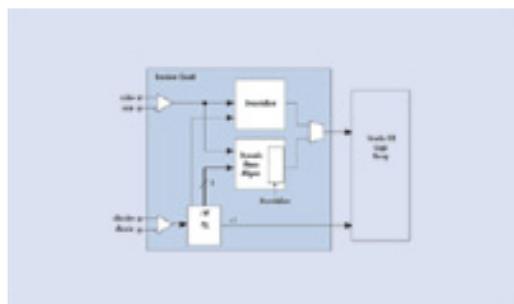
[3]

The transmit channel-to-channel skew, including parasitic skew, data jitter, data duty-cycle distortion and PVT effects, is estimated at 400 ps. The clock duty-cycle distortion and PLL output jitter are estimated at 25 ps and 20 ps, respectively. Finally, the sampling window, including parasitic skew, clock skew, sampling error and data-dependent jitter is estimated at 440 ps. Taken together, these elements allow a tolerance of 270 ps of board skew, or 135 ps on either side of the sampling window. The typical amount of board skew and jitter, earlier estimated in the range of 144 to 384 ps, is clearly outside the tolerable range in which this system could reliably operate.

Dynamic Phase Alignment

DPA technology has been developed to address the inadequacies of static phase-alignment methods. The goal of DPA is to allow devices to actively respond to changes in the operational board skew. Devices equipped with DPA continuously check the incoming data and adjust the phase of the clock to align with it. Several industry standards responsible for defining chip-to-chip interfaces, including SPI 4.2, have recognized the value of DPA, and have included or recommended it in their specifications.

An example of a dedicated DPA implementation is seen in the device in Figure 4. Currently, this is the only instance of a hard-silicon DPA implementation combined with programmable logic. Figure 4 shows how DPA is implemented in these devices in the form of a dynamic phase-aligner block.



[4]

As shown in Figure 4, the serial data arrive via the rx_{in} pins and is routed to either the dynamic phase-aligner block or directly to the deserializer block, bypassing the DPA circuitry. These channels can operate in either DPA mode or non-DPA mode; choice of mode is determined by the user and is set during device configuration. The receive clock arrives via the $clk_{rx_{in}}$ pins and is used by the PLL to generate eight phase-shifted versions of the clock for the dynamic phase-aligner block each 45 degrees out of phase. The PLL also generates multiplied or divided versions of the receive clock for the deserializer block. The dynamic phase-aligner block uses the incoming data and clocks to generate a new clock that is phase-aligned with the data. Clock domain transfer and word alignment are then performed in the synchronizer and data realigner. The data are then deserialized by factors of four, eight or ten, depending on the system requirements, before being sent to the logic array.

DPA circuitry relies on transitions in the data stream to adjust the clock phase. During long run lengths in which the data do not change, the DPA circuitry remains locked on the last chosen phase. These long runs have an adverse effect on the DPA circuitry's ability to accurately adjust the clock in response to changing skew. To prevent this, several chip-to-chip interface standards specify a maximum run length for data. For example, with the SPI-4.2 specification, the maximum run length without data transitions is 4,096 clock cycles, while the RapidIO standard specifies a maximum of 64.

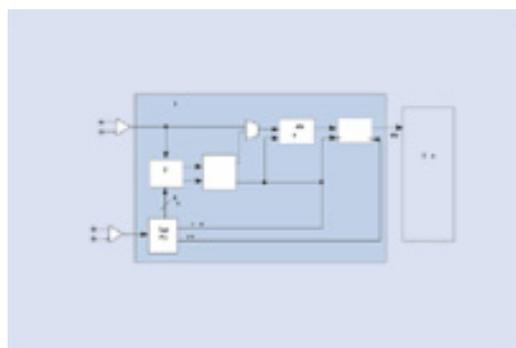
Benefits of Built-in Dynamic Phase Alignment

A complete, FPGA-integrated hard-silicon DPA solution offers several benefits to system designers. It is implemented for each data channel, such that each channel receives its own phase-adjusted clock. This individual alignment for each channel minimizes the chance for errors introduced by mismatches in signal propagation paths. Also, it does not require a training mode; rather, it continuously realigns the clock to the data during device operation. The training patterns specified by

The Need for Dynamic Phase Alignment in High-speed FPGAs

Published on Wireless Design & Development (<http://www.wirelessdesignmag.com>)

standards such as SPI-4.2 or RapidIO are supported, but no training pattern is required when using DPA with other interfaces. This continual operation means that long-term changes in skew or jitter characteristics that arise from temperature or other variations are taken into account. This is a key differentiation between a true silicon DPA solution and a soft DPA implementation embedded into FPGA logic. Additionally, DPA does not require reconfiguration for recalibration, as static phase-alignment does. Finally, a hard DPA solution preserves valuable clock, PLL and logic resources that would be implemented in a soft DPA solution. This combination of reliability, high data rates, and a low-risk, low-investment implementation makes hard-silicon DPA an important feature for high-speed FPGA designs.



[5]

A complete DPA solution for FPGAs handles distribution of multiple clocks on the board, allowing several synchronous links from a single clock, which eases board layout and saves engineering time and costs. Without DPA, the use of high-density pin-out packages such as BGA packages in high-speed boards for source-synchronous functions often requires additional board layers to match the trace lengths. Also, by including word-alignment logic in the DPA block, users need not consume internal logic resources to build this function. For high performance FPGA architectures, this function would otherwise consume 100 to 200 LEs per channel. A complete, built-in DPA solution, including the word-alignment circuitry, aligns the received data correctly even if the traces vary in length.

Conclusion

The inclusion of integrated DPA circuitry in FPGAs relaxes board layout restrictions associated with static phase-alignment methods. By eliminating the need to match trace lengths and layout boards manually, DPA reduces engineering effort, shortens time-to-market, and lowers labor costs, which reinforces and maintains the traditional strengths of FPGAs for data-intensive applications. DPA also allows more efficient board usage and adds design options by simplifying the use of daughter cards and connectors in high-speed, timing-sensitive systems. With the ability to increase design flexibility and significantly shorten development times, integrated DPA is a necessity for FPGAs targeted at high-data-bandwidth applications.

The Need for Dynamic Phase Alignment in High-speed FPGAs

Published on Wireless Design & Development (<http://www.wirelessdesignmag.com>)

Note: For more information on how integrated DPA is implemented in Stratix GX FPGAs, see AN 236: Using Source-synchronous Signaling with DPA in Stratix GX Devices.

Source URL (retrieved on 04/01/2015 - 2:56am):

<http://www.wirelessdesignmag.com/product-releases/2006/02/need-dynamic-phase-alignment-high-speed-fpgas>

Links:

[1] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0602/wd62feature1_1_lrg.jpg

[2] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0602/wd62feature1_2_lrg.jpg

[3] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0602/wd62feature1_3_lrg.jpg

[4] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0602/wd62feature1_4_lrg.jpg

[5] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0602/wd62feature1_5_lrg.jpg