by Brock J LaMeres

FPGA

when to go

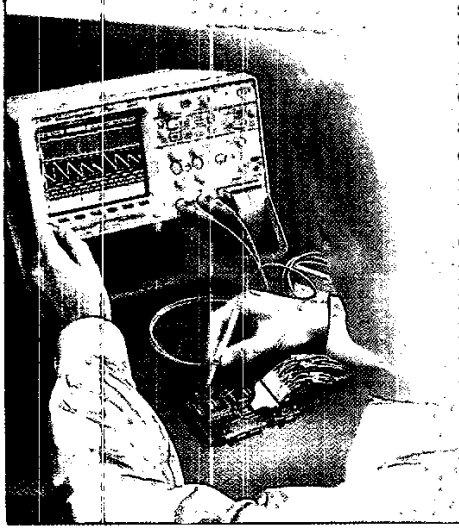
FIELD-PROGRAMMABLE GATE ARRAYS NOW INCORPORATE HIGH-SPEED SERIAL INTERFACES. BUT WHICH ARE THE RIGHT TYPES OF INTERFACE AND CLOCKING SCHEME TO USE?

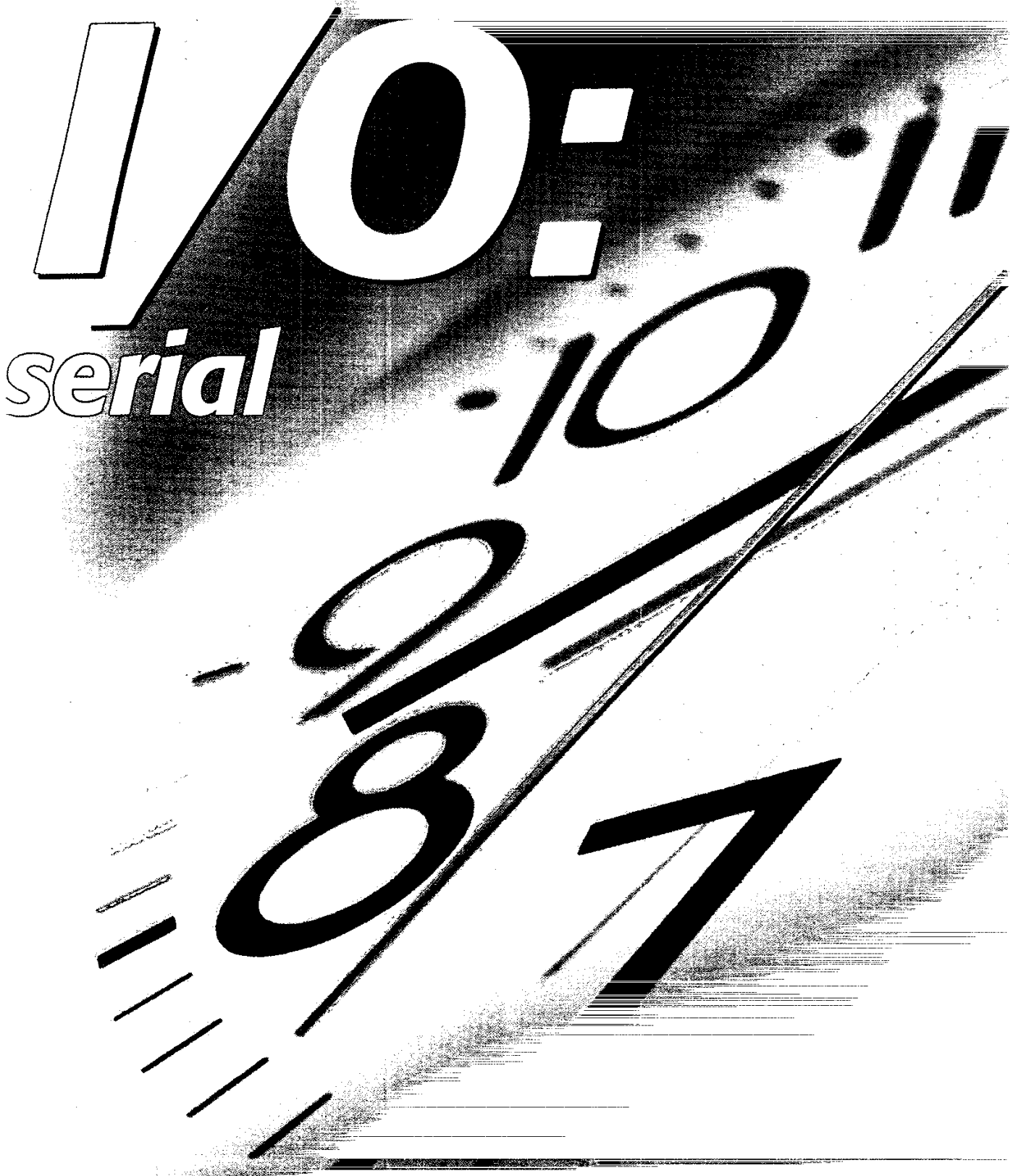
Over the past decade, field-programmable gate arrays (FPGAs) have gained a foothold as some of the most used building blocks in digital systems. The flexibility of an FPGA allows designers to decrease hardware design cycles while adding inherent feature upgradeability in the final product. In addition, the data rates of modern FPGAs are competing with application-specific integrated circuits, thus allowing the needed system performance to be achieved using what was once only a prototyping vehicle.

The data rates of modern FPGAs are giving designers the freedom to create their own application-specific buses. However, designers are quickly learning the pitfalls of running I/O at high speeds. Factors such as channel-to-channel skew, jitter, and aperture window size are limiting the theoretical data rates of the FPGA's specifications. To address these issues, FPGA system designers are following suit to their ASIC predecessors and adopting I/O architectures that inherently reduce the effect of the above-mentioned factors.

A number of clocking schemes are available to designers and are shown in Fig 1. Synchronous clocking is the most widely used clocking scheme in digital systems. In this type of clocking, there is one clock that is distributed to all synchronous logic in the system. All transactions occur at a particular edge of this one clock. This type of clocking has the advantage that it is relatively simple to implement. However, in order to ensure timing accuracy, the clock distribution network must be matched in electrical length to all of the synchronous logic in the system. This becomes increasingly difficult when clock paths enter FPGAs with complex logic and connect multiple FPGAs that are created with various processes and different packaging. The variation in integrated circuit processes and construction leads to timing error that degrades the performance of this type of clocking. Another limitation is that after a clocking event, the next clock must wait until the data has successfully travelled from one FPGA to the other. Examples of this type of architecture are the PCI bus and interfaces to synchronous memory.

Source-synchronous clocking was developed to address the difficulty that synchronous clocking has with matching the →




I/O:
serial

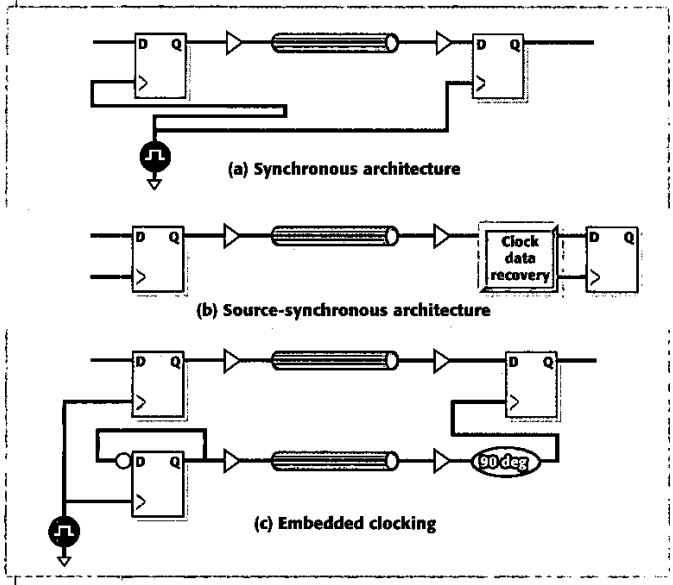


Fig 1: Three architectures for serial interfaces

Individual Channel Data Rate	Clocking Scheme
DC – 300Mb/s	Synchronous
300Mb/s – 2Gb/s	Source Synchronous
>2Gb/s	Embedded Clock

Table 1: Per-channel data rates observed in industry for various clocking architectures

$$UI = t_{Symbol}$$

$$DataRate = \frac{1}{UI}$$

$$Throughput = (\#ofChannel) \cdot (DataRate)$$

Fig 2: Achievable unit interval calculation

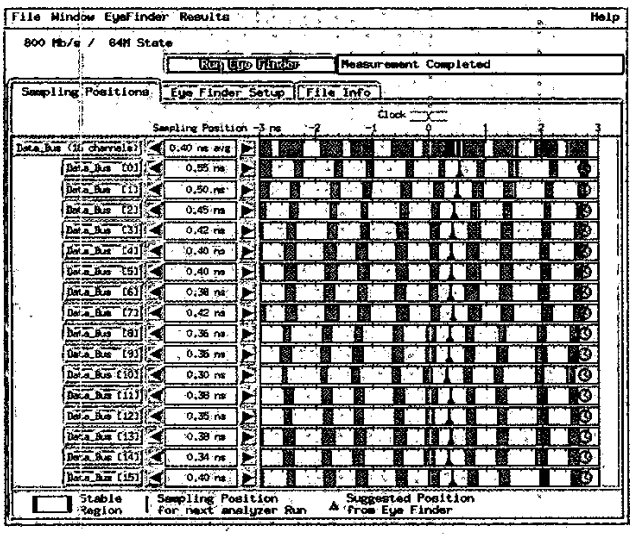
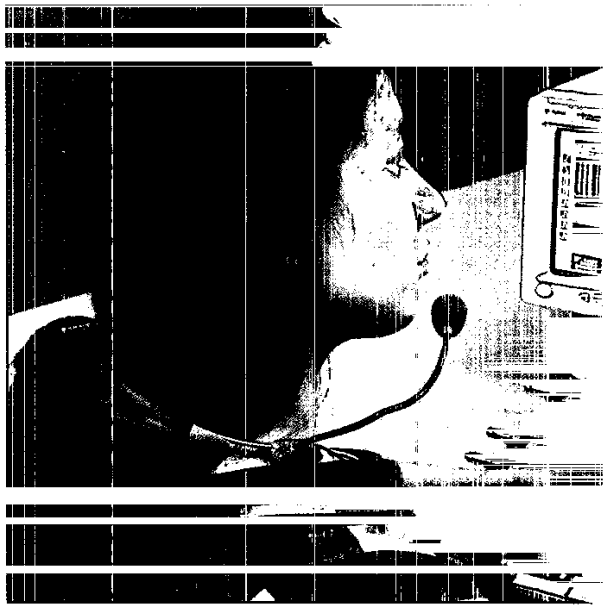
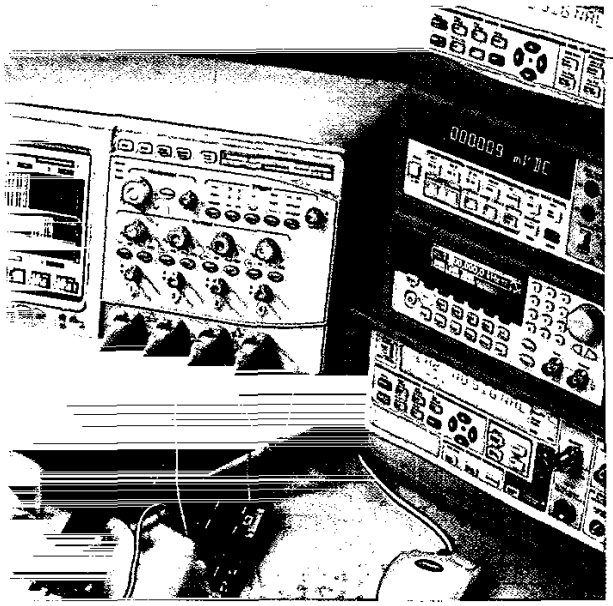


Fig 3: Channel-to-channel skew picked up by an 'eye-finding' tool



clock distribution paths to each element in the system. In this type of clocking, the circuit producing the data pattern will create its own clock, which is transferred along with the data. Generating the clock in the same geographical location as the data, in addition to having the clock traverse the same media as the data bus, creates a much tighter timing correlation. The tighter timing means that the data-valid windows of the data being latched are better aligned with the clock used to latch them. This reduced channel-to-channel skew allows data transfer rates to exceed that of a synchronous architecture. Examples of this type of architecture are the front-side bus for Intel's PC processors and double data-rate memory interfaces.

Source-synchronous clocking is prone to the same channel-to-channel skew as synchronous timing, albeit at a much higher data rate. The difference in process variation and interconnect structure between channels within an FPGA will contribute to channel-to-channel skew no matter how tightly the clock is coupled to the multiple signals. To address this problem, embedded clocking was invented. In embedded clocking, the data is encoded in a manner that will guarantee a certain number of transitions per time. An example is 8B/10B encoding. By ensuring the data will transition a certain percentage of unit time, a phased-locked loop (PLL) can lock to the data stream. With embedded clocking, a PLL is used at the receiver that will lock onto the incoming data stream and create a clock based on the transition frequency. This clock is then centred within the data valid window. The drawback of this is that the clock centring is only valid for the data channel to which the PLL has locked. However, the data rate for an individual channel is only limited by its jitter margin and aperture window. This type of clocking can achieve extremely high data rates, of more than 2·5Gbit/s, compared with synchronous and source synchronous designs. This type of design requires more complex circuitry at both the driver and receiver but the data rates achieved are so much higher than previous architectures that the channel counts can be reduced while still delivering an overall throughput increase.



The total throughput, or digital bandwidth, is defined as the number of symbols that can be transmitted per second. This depends on the unit interval that can be achieved per channel and the number of channels used in the I/O architecture, as shown in Fig 2. Channel-to-channel skew refers to the time difference of the data valid windows between various signal paths. This issue is due to electrical length differences in the physical signal paths. These paths vary for each channel due to the implementation of the circuit.

The skew between channels will limit the data rate of the I/O architecture. This is because the synchronising clock for the data signals must be placed such that it can successfully latch in all of the data channels in the bus. To achieve this placement, the net data-valid window of all of the overlaid channels must be large enough for the clock to be placed within. The more channels used, the more the chance of channel-to-channel skew. As more channels are used, the net data-valid window will decrease, thus reducing the maximum transfer rate.

There are two main sources of skew. The first is the electrical length mismatch on the IC. The majority of this mismatch comes from the packaging interconnect. When using an FPGA, the complex logic can also be a major factor. As CMOS dice approach sizes of 20×20mm and packages have signal counts reaching into the thousands, the difference in channel path length becomes considerable. The on-chip skew is also exacerbated by the fact that signals are being driven from one large FPGA package to another. This can effectively double the skew due to on-chip path length.

The second source of skew is due to channel mismatch on the PCB. This skew is due to either a physical mismatch of PCB traces on the same layer or a propagation delay mismatch of PCB traces on different layers. Fig 3 shows an example of channel-to-channel skew picked up using an 'eye-finding' tool on a logic analyser made by Agilent Technologies.

Jitter is a term that describes the timing uncertainty within a unit interval. Put another way, jitter is the amount of time within a unit interval in which it cannot be guaranteed that the data is at a stable logic level. This uncertainty region will limit how small a unit interval can be used while still transmitting reliable data. The two major subsets of jitter are deterministic and non-deterministic jitter.

Deterministic jitter refers to sources of jitter that can be calculated. For example, if a product is specified to operate with 5% power supply variation, then the timing uncertainty due to supply drop needs to be considered in the data-rate specification. While how much timing uncertainly will vary from application to application, the entire range of timing uncertainty must be accounted for. Some possible sources of deterministic jitter can be process variation, inter-symbol interference, reflections, simultaneous switching noise, power supply droop, and resistance-capacitance load variation. These sources can all be quantified in the design and summed to produce a timing-uncertainly total. The worst-case impact of each of these sources must be accounted for in a stable design.

Non-deterministic jitter refers to the statistical sources of timing uncertainty. These are sources in which the noise contribution is modelled by their probability distribution rather than their worst-case values. Traditionally these sources are modelled using a Gaussian distribution. The figures-of-merit for these sources are their root-mean-square value or standard deviation. For these types of sources, the worst-case impact does not need to be accounted for as, statistically, it will happen infrequently. Instead, a bit error rate is used to predict system stability. A few examples of these types of noise sources are thermal noise and the shot noise caused by charge carriers moving through a semiconductor. Multiple sources of statistical noise are accounted for using a root-mean-square summation to find the net contribution of statistical noise.

Jitter can be very difficult to predict. Typically its contribution is found using jitter measurement [illegible] in an oscilloscope. By allowing an [illegible]

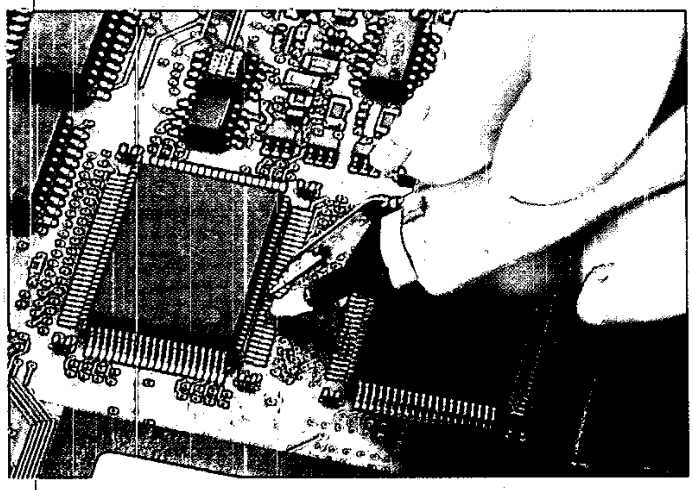
accumulate for a relatively large amount of time, the distribution of jitter can be found. A note on this technique is that all of the sources of jitter are measured simultaneously. Although this has the drawback of not isolating individual sources, it is usually the only method available to FPGA designers. An example is shown in Fig 4.

The final contribution to the speed of an I/O design is the aperture window of the receiving element. This is traditionally called setup-and-hold but refers to the minimum time that the data must be stable before and after the timing event in order for the receiver to capture the symbol. The aperture window must be able to fit within the net data-valid window of all of the overlaid data channels to ensure successful data collection. FPGA manufacturers will specify this value. However, complex logic and data path variation will degrade the ideal specification. In most cases a measurement is needed.

The combination of channel-to-channel skew, jitter, and receiver aperture dictates how fast the I/O design can operate. We define the channel-to-channel skew as the time difference between the transition regions of two channels when operating in the same phase, that is, when those channels are designed to transmit a symbols at the same time. The equations in Fig 5 relate the total bit transfer rate per channel to the sources of error described above.

The first step in selecting an I/O architecture is determining the design's data-throughput requirement. Once this is defined, I/O architectures can be evaluated for selection. This is an interactive process requiring measurement data. All of the factors, such as channel-to-channel skew, jitter and aperture window, will increase as signals are added to the I/O design. The best way to find these contributions is through empirical data from oscilloscopes and logic analysers.

Starting with the simple synchronous architecture, signals can be added to increase the overall throughput. As signals are added, the degradation factors must be observed.



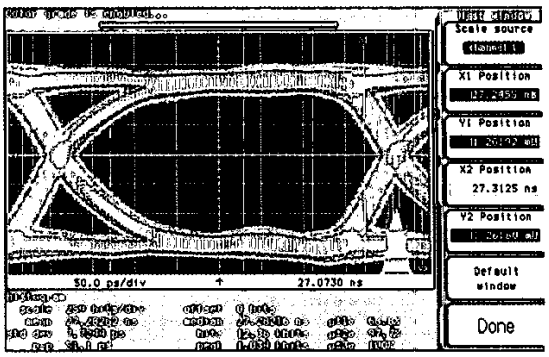


Fig 4: Jitter measurement with histogram showing jitter distribution

$$UI \geq t_{chan-to-chan} + t_{jitter} + t_{Apperature}$$

$$Throughput = \left(\frac{\#ofChannels}{UI} \right)$$

Fig 5: Relationship between bit transfer rate and error

At some point, the addition of signals is no longer practical because of the adverse signal integrity contributions, the main contribution being channel-to-channel skew. At this point, the source synchronous architecture can be explored. The major advantage of this architecture is that it dramatically reduces the channel-to-channel skew within a data group. At some point the channel-to-channel skew will again become an issue, but at a much higher data rate. The skew in this architecture will be on the same order of magnitude as the jitter contribution.

Finally, the embedded clock architecture can be examined. This architecture has the advantage in that there is no channel-to-channel skew due to the clock/data recovery design. In addition, the jitter is reduced because there is no longer a data channel and a clock channel that possess jitter. This architecture will be limited only by the jitter and aperture window contribution. If industry I/O standards are observed, they typically fall into the categories shown in Table 1.

As FPGA speeds have increased over the past decade, designers are using FPGAs as an alternative to ASICs. However, designers are quickly discovering that the increased FPGA speeds are leading to the same signal integrity problems that ASIC designers have been facing for years. As signals are added to an I/O architecture to increase the total throughput, the channel-to-channel skew, jitter, and aperture window contributions are increased. To address these problems, FPGA designers must turn to different clocking architectures. In order to maintain the trend of increased system data rates, a designer using FPGAs must first understand all of the contributions to the unit interval reduction in order to be successful. ■

Brock LaMeres is a hardware design engineer for Agilent Technologies.