XAPP225 (v1.1) April 4, 2002

Data to Clock Phase Alignment

Author: Nick Sawyer

Summary

When designing digital systems, there is often a requirement to synchronize incoming data and clock signals with an internal system clock, i.e., the internal and external clock are at exactly the same frequency, but due to variable backplane, board, or application-specific standard product (ASSP) delays, the phase relationship is not known. The circuit described in this application note addresses this issue for both single traces and data busses up to 160 MHz in a Virtex™-E, -7 device. The speed limitation is imposed by the maximum frequency that can be accepted by the Data Locked Loop (DLL), in a mode where it is capable of providing both a new clock and a new clock shifted by 90 degrees. The maximum speed would be 210 MHz in a Virtex-II, -5 device or a Virtex-II Pro™, -6 device. A typical application is shown in Figure 1.

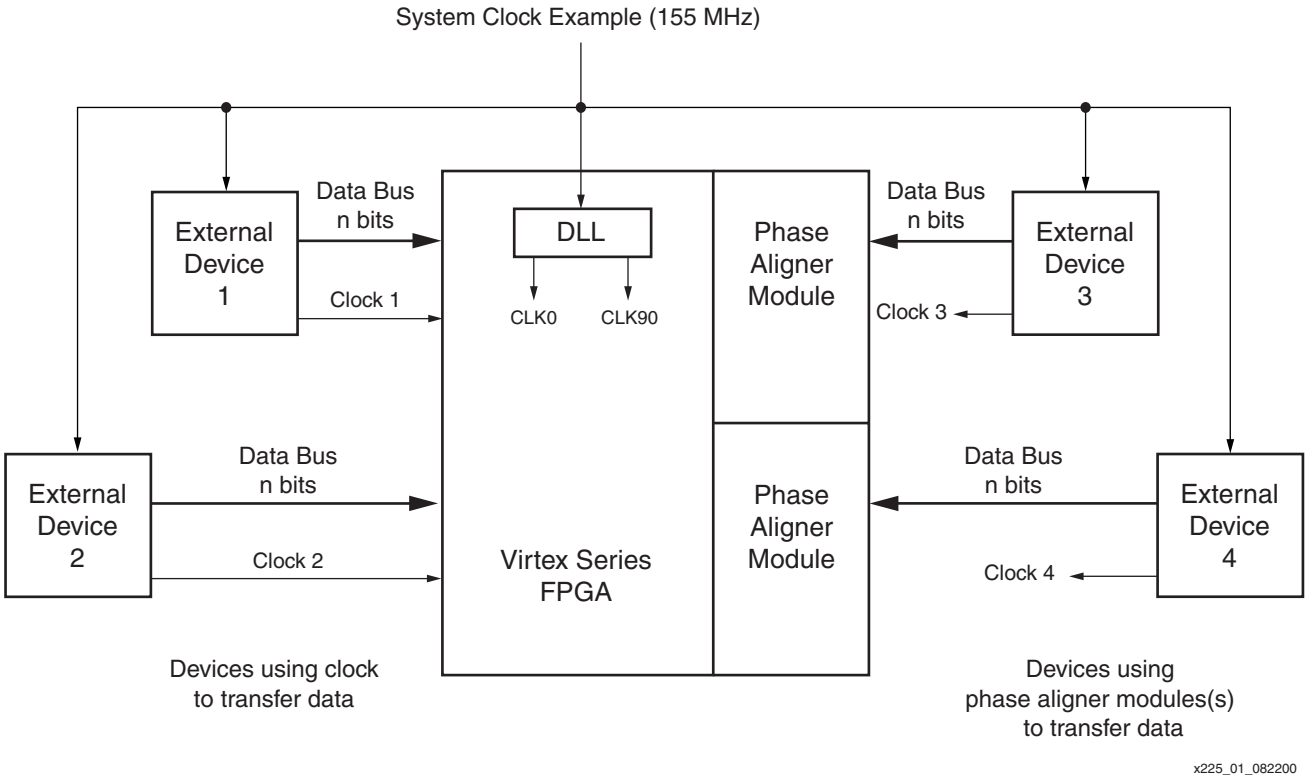

Figure 1: **Typical Application of Data to Clock Phase Alignment**

Introduction

There are two classical methods of achieving synchronization. One is to sample the data with four different clocks and decide which of the four possible clocks is most valid. A multiplexer is used to provide the most valid clock to the rest of the required logic running at the communications link speed. Since there are only four global-clock buffers in a Virtex device,

they would be used up immediately with this method. In addition, inserting a multiplexer in the clock path effectively disables the DLL clock zeroing function. The Virtex-II series is improved with sixteen global clock buffers. However, using a global clock buffer for each incoming clock can be an inefficient use of system resources at low clock speeds.

The other method (described in detail here) is to sample the data on four phases of the same clock, decide which phase is the most "valid", and then resynchronize the data to the system clock. This method is more resource efficient, requiring only two global buffers, one DLL or one DCM.

The incoming system clock is fed to a DLL component, and the DLL CLK0 is used to provide a clock (CLK) for the synchronizer circuit, as well as feedback for the DLL. There is also a version of the input clock delayed by 90 degrees (CLK90), and synchronized with the original clock. These waveforms are shown in the Figure 2 timing diagram along with the four possible data arrival cases used in the next section.

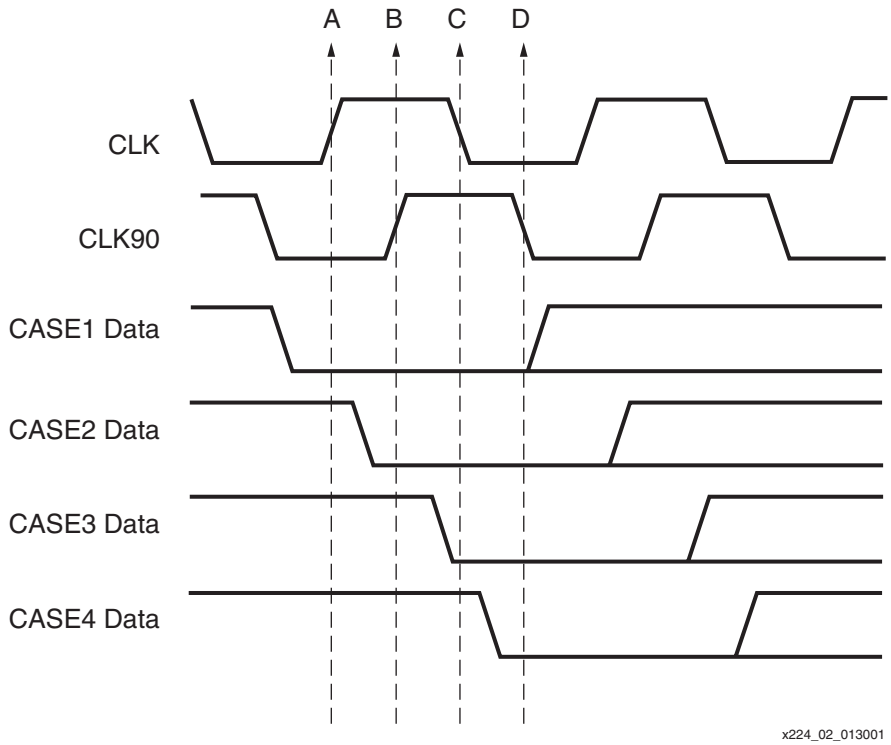

Figure 2: **Timing Diagram**

As shown in Figure 3, the incoming data is applied to four flip flops, two clocked by CLK (one rising edge and one falling edge), and two by CLK90 (rising and falling edges). It is important that the delay from the input pin to these four flip flops be almost equal. This is easily achieved by giving the software a MAXSKEW parameter for this net, of 500 ps, for example. The absolute delay is irrelevant; only the skew is important. However, when dealing with data busses, it is also prudent to apply a MAXDELAY parameter of approximately 1.0 ns to each net in the bus.

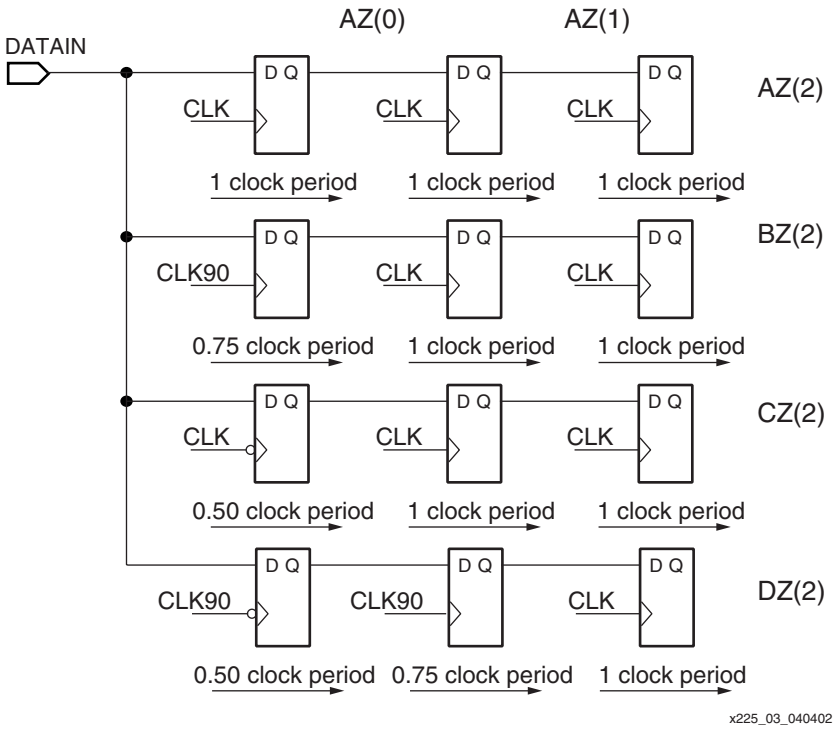

Figure 3: **Input Stage**

The first flip flop is clocked by the rising edge of the clock described as time domain A. The second flip flop is clocked by the rising edge CLK90 (time domain B); the third flip flop is clocked by the falling edge of CLK, (time domain C); and the fourth is clocked by the falling edge CLK90, (time domain D). As shown in the timing diagram (Figure 2), this gives four data sample points, each separated by 90 degrees of the original clock frequency. In the case of a 160 MHz system clock, this logic is effectively running at 640 MHz.

These four sample points are then clocked once more, to remove any metastability issues and to move them into the same time domain. This actually takes place in two stages (again to avoid any four times clock frequency logic paths).

In the decision stage, shown in Figure 4, the circuit detects transitions on the data lines. The signals AAP to DDP recognize positive transitions, and the signals AAN to DDN recognize negative transitions. Eight signals are now available for the decision process. Four mutually exclusive signals can now be decoded, where only one transitions High whenever there is a data transition. These four conditions are as follows:

1. AAP = BBP = CCP = DDP = 1, or AAN = BBN = CCN = DDN = 1. Time domain A was the first to see the data. Therefore, the data from C is used for forwarding.
2. AAP = 1 and BBP = CCP = DDP = 0 or AAN = 1 and BBN = CCN = DDN = 0. Time domain B was the first to see the data. Therefore, the data from D is used for forwarding.
3. AAP = BBP = 1 and CCP = DDP = 0, or AAN = BBN = 1 and CCN = DDN = 0. Time domain C recognized the transition first. Use the data clocked in during time domain A for forwarding into the system. This is the data that has been sampled midway through its period, i.e., the best noise margin.
4. AAP = BBP = CCP = 1 and DDP = 0, or AAN = BBN = CCN = 1 and DDN = 0. Time domain D was the first to see the data. Therefore, the data from B is used for forwarding.

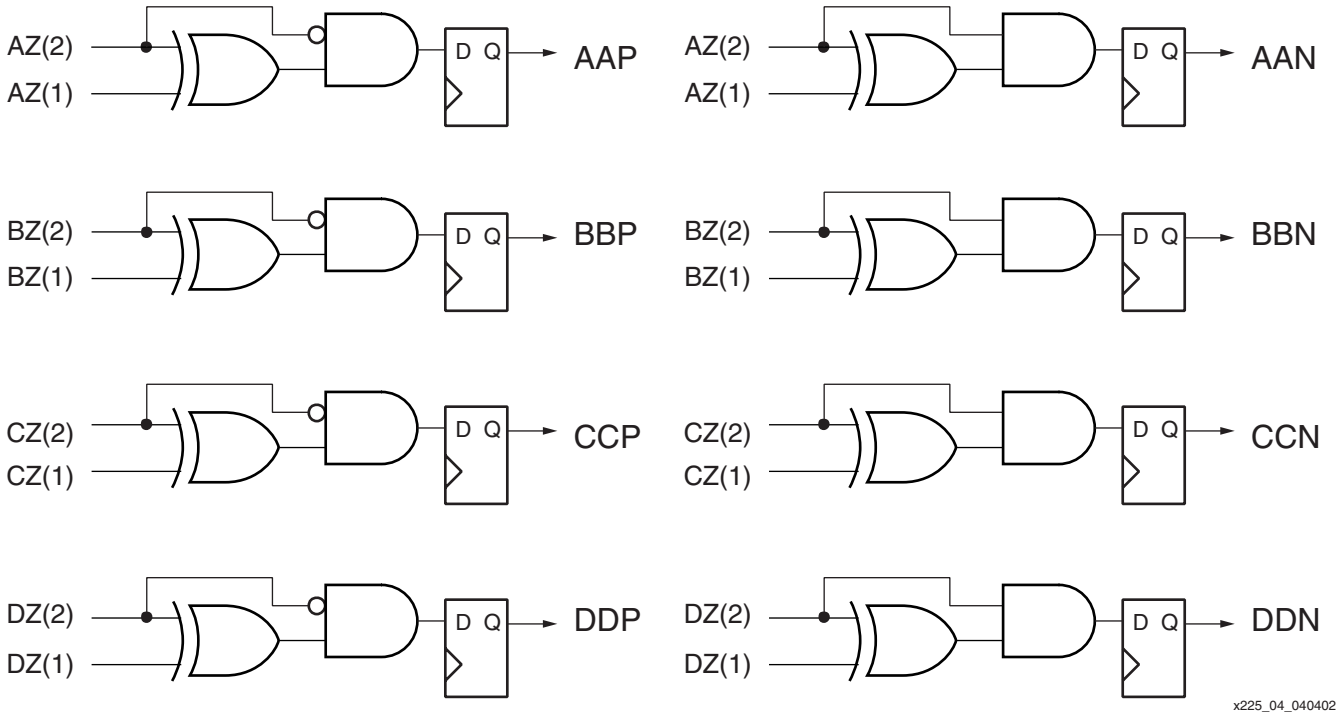

Figure 4: **Decision Stage**

The selection of data is done with a very simple multiplexer used to select data from the appropriate time domain (Figure 5). The complexity increases, in the real world, since the data is not fixed in time, but moves slightly with temperature and voltage changes. That is, it is quite possible that the circuit will synchronize with one time domain, but will resynchronize later to another. At 160 MHz, this implies that the data could have "moved" at least 1.6 ns. This also illustrates another use for the circuit, in bus systems, where clock delays will change as boards are added and removed. The circuit will automatically compensate for these delays.

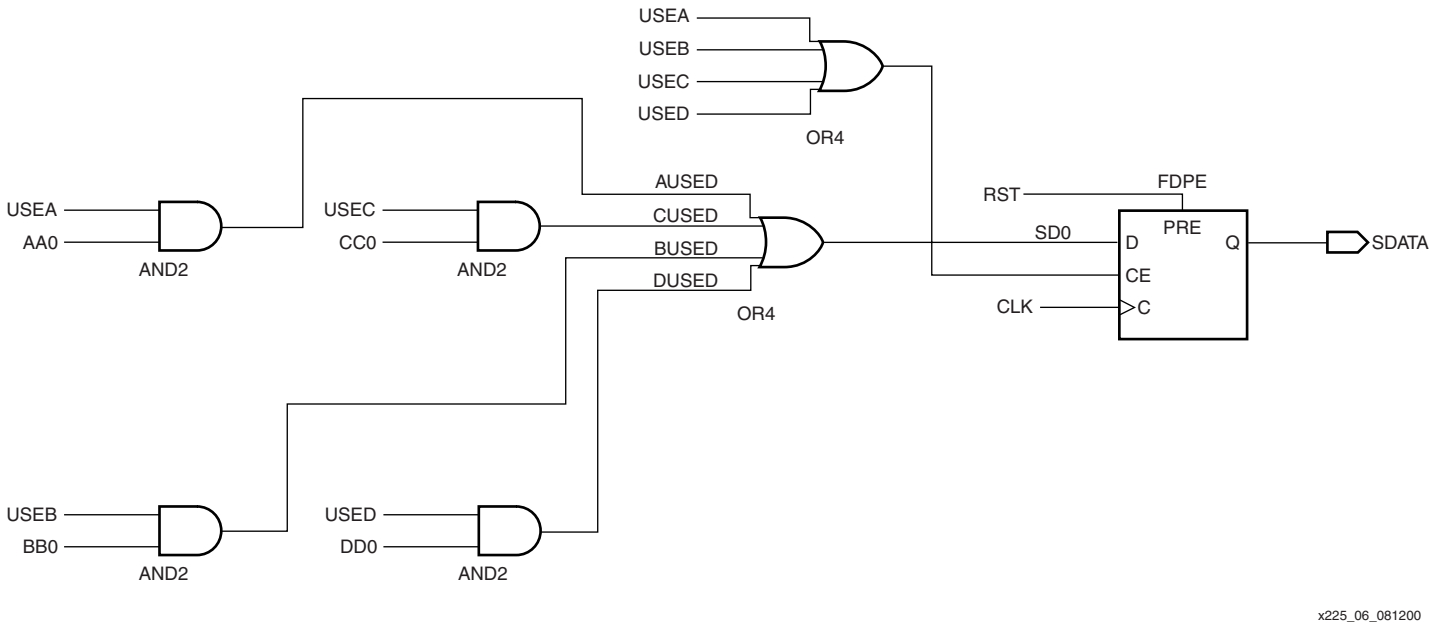

Figure 5: **Data Multiplexer**

As the data "moves", the multiplexer is used to select the data from one of the four available time domains, with one exception. If the data moves from domain A to domain D, then effectively one stage of delay to the circuit is added, i.e., a "wraparound" occurs. This is done by detecting this transition (and also the opposite, i.e., going from D to A) and using a variable length shift register. This is easily achieved by using the SRL16 primitives in the Virtex series. When the circuit starts up, the SRL16s are programmed to be a 2-bit delay, if the transition mentioned above (A to D) is detected, then the SRL16s are reconfigured to be a 3-bit delay, or a 1-bit delay for D to A (Figure 6).

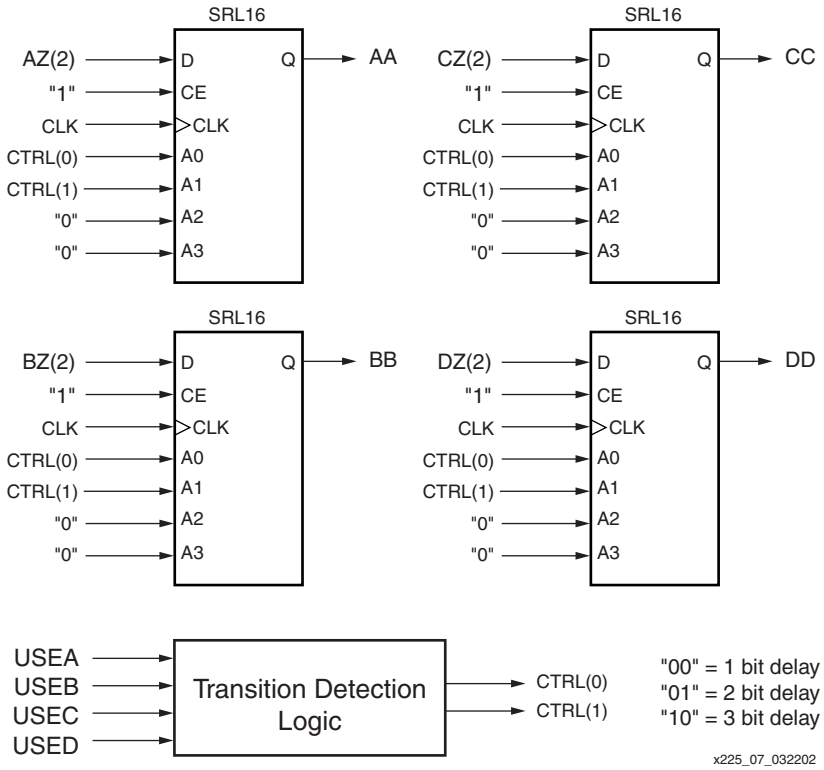

Figure 6: **Variable Length Shift Register**

If no transition has occurred, the circuit is locked, i.e., the selection logic does not change state, and if the transition wanders into another phase, then the selection logic is changed appropriately, as described above.

Metastability

Since it is possible that the data will not properly change state between clock sample points, there is a possibility of metastability. As the data transition approaches a sample point (for example, clock sample A in Figure 2), it will eventually end up inside the setup time to the clock CLK, causing one of several outcomes to occur.

If the flip flop is fast enough to still see the transition, then AAx = BBx = CCx = DDx = 1, (case 1) and all will work as previously described. If the flip flop does not see the transition, then AAx = 1 and BBx = CCx = DDx = 0, (case 2) and again all will work properly.

Finally, the flip flop could briefly enter a metastable state. If this occurs, then the second synchronizing flip flop will still "see" a 1 or 0 and will register that state, leading to the same arguments as above. There is no problem as long as the load on the potentially metastable flip flop is one, except when the metastable period is *exactly* equal to the input clock period. However, this event is extremely unlikely with today's very fast silicon, and even then, there are three further registers in the data path. Therefore, the chance of a metastable event upsetting the operation of the circuit is vanishingly small.

Conclusion

Virtex series devices can be used to interface to external components running off the same clock but with an unknown phase relationship. This phase relationship can also be allowed to vary up to plus-or-minus one clock cycle in operation without affecting data integrity. Data transfers can therefore occur at speeds up to 210 MHz, without having to use a clocked mechanism, which would waste clock resources, when using the phase aligner described in this application note.

Reference Design

The reference design circuit is implemented in HDL. It is fully synthesizable and comes in two variations. One variation is a master unit (sync_master.vhd), which will deskew one or two data lines and provide the control signals for further slave units. The slave modules (sync slave.vhd) also deskew two data lines and because all the clocks are on global buffers the slave modules can be added to a design ad infinitum.

The reference design files (**xapp225.zip**) include a top.ucf file, containing all the timing constraint information. It is important to use this file, because some paths are very fast. Of course, in the case of busses, the data input delays have to be very closely matched. The best way to achieve this is to locate the bus input pins closely together.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/18/00	1.0	Initial Xilinx release.
04/04/02	1.1	Revised for Virtex-II Series devices.