XAPP224 (v2.5) July 11, 2005

Data Recovery

Author: Nick Sawyer

Summary

Data recovery allows a receiver to extract embedded clock data from an incoming data stream. The receiver usually extracts the data from the incoming clock/data stream and then moves this data into a separate clock domain. Sometimes, the receiver's clock is also used for onward data transmission. The circuit described in this application note provides a partial solution at data rates up to 160 Mb/s in a Virtex™-E -7 device and a Spartan™-IIE -6 device, up to 320 Mb/s for a Spartan-3 -4 device, and up to 420Mb/s in a Virtex-II -5 device or a Virtex-II Pro -6 device. The solution is partial in the sense that no clock is actually recovered, but the data arriving is fully extracted. The speed is limited by the maximum frequency that can be accepted by the Data Locked Loop (DLL), in a mode where the DLL is capable of providing both a new clock, and another clock shifted by 90 degrees. A typical application is shown in Figure 1.

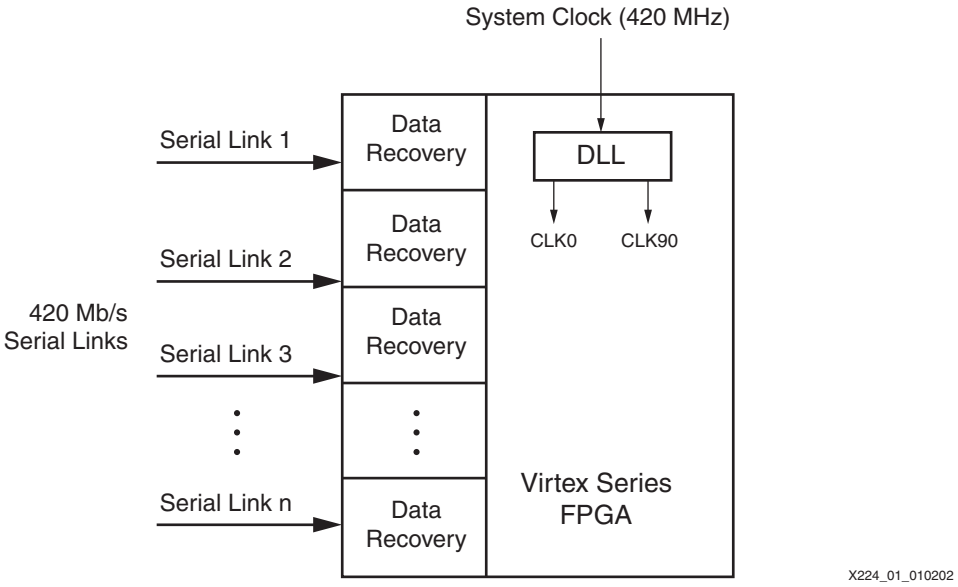

Figure 1: **Typical Data Recovery Application**

Introduction

The circuit described herein uses a clock (local oscillator) that is running at the same nominal frequency of the data stream being decoded. Typically, this means that the local oscillator is either slightly faster or slightly slower than the incoming clock/data stream. For example, a typical link may be running at 400 MHz plus or minus a small but different variation. The actual performance relative to the clock rate will be discussed after giving a description of how the circuit works.

Assuming that the incoming clock/data line is not encoded beyond a 1 being transmitted as line High and a 0 as line Low. Minimum transition requirements are discussed in the following sections.

For the Virtex-E device or for a slow link using a Virtex-II device, 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. Another version of the input clock delayed by 90 degrees (CLK90) and synchronized with the original clock is available. For a fast link using the Virtex-II device, two DLL (DCM) modules are required (see Figure 2); one provides the CLK and the other provides CLK90, using the fixed phase shift feature available to the Virtex-II device.

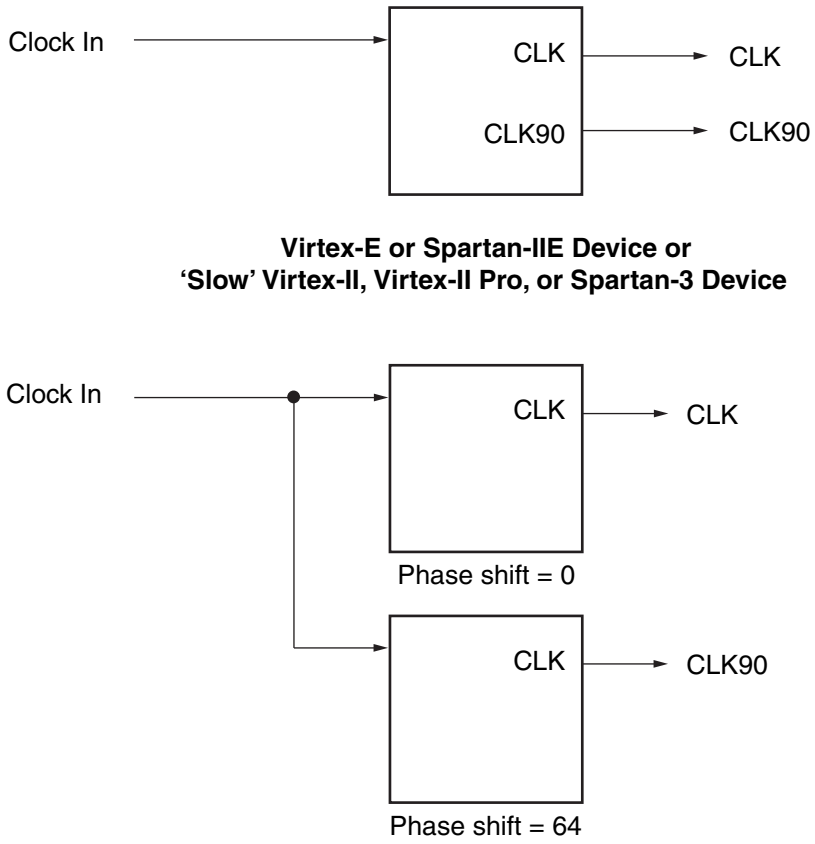

Figure 2: **Clock Generation Using One or Two DLLs/DCMs**

These waveforms are shown in the Figure 3 timing diagram along with the four possible data arrival cases used in the next section.

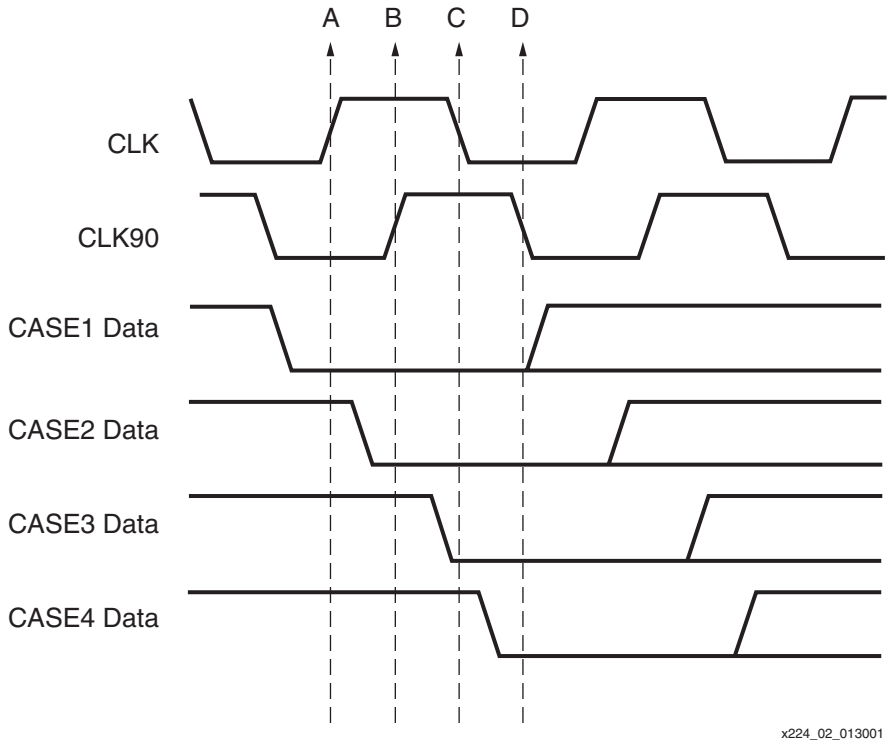

Figure 3: **Timing Diagram**

As shown in Figure 4, 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.

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 on the falling edge of CLK (time domain C); and the fourth is clocked on the falling edge CLK90 (time domain D). As shown in the timing diagram (Figure 3), this gives four data sample points, each separated by 90 degrees of the original clock frequency. In the case of a 420 MHz system clock, this logic is effectively running at 1680 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 or three stages (again to avoid any four times clock frequency logic paths).

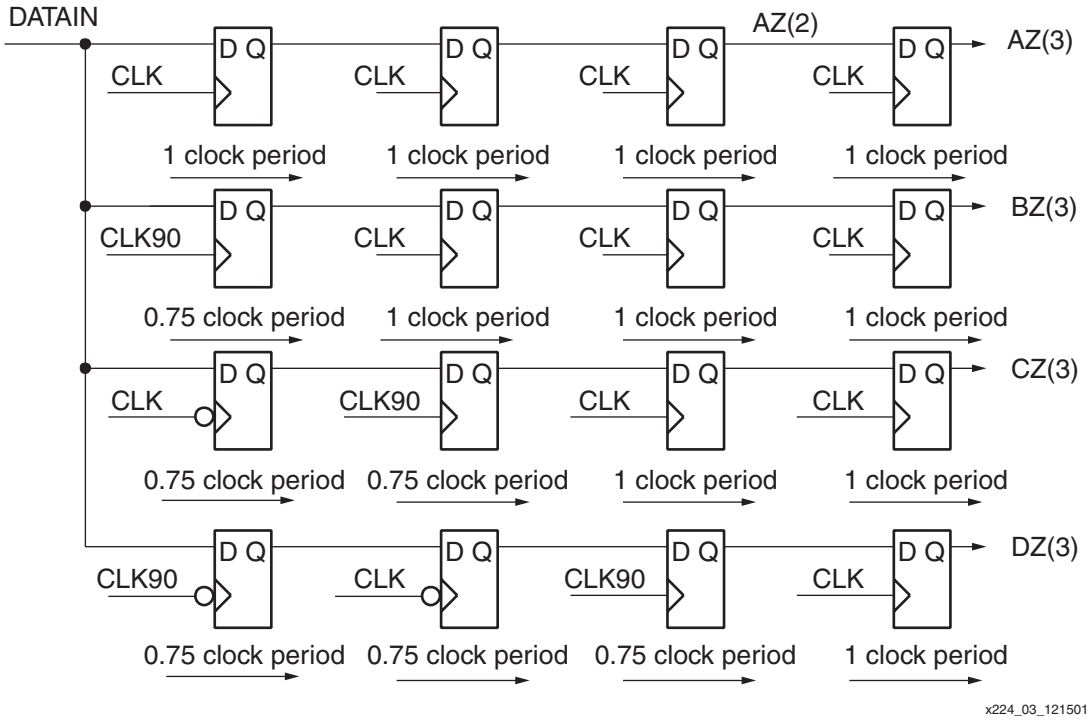

Figure 4: **Input Stage (Four Stage Version)**

In the first decision stage, shown in Figure 5, 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.

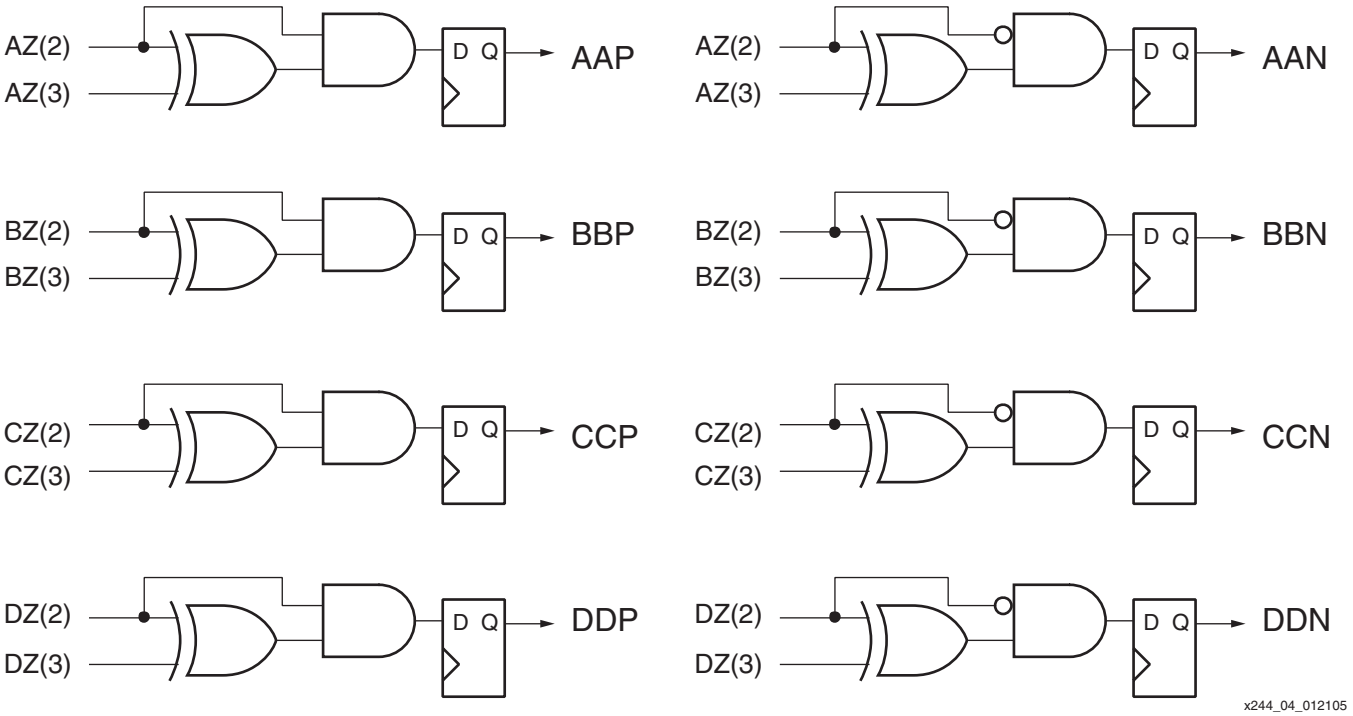

Figure 5: **Decision Stage 1**

The selection of data is done with a very simple multiplexer used to select data bits from the appropriate time domain, i.e., if the circuit has decided to use data from time domain A, then data bit AZ(3) is passed on to the output.

As discussed, the local clock will probably be a bit faster or slower than the incoming clock/data stream, and therefore, there will be clock cycle where the received data is either invalid (local clock faster) or two data bits will be required (local clock slower). As the clock/data *moves* in time, the multiplexer can be used to select data from one of the four available time domains, but with two exceptions. If the circuit moves from domain A to domain D, then the data being output is actually invalid. This is indicated to the user by lowering the data valid (DV) signal to 0. If the circuit moves from domain D to domain A, two bits of data need to be generated. This is indicated to the user by setting both data valid bits High. This is shown in Figure 6. In an actual system, these events occur continuously but randomly, as the circuit moves due to clock jitter and data eye closure, but the data output will always be valid according to the data available bits. For example, with a 401 MHz (2.506 ns period) local clock and an incoming data stream of 400 Mb/s (2.500 ns period), the received data will be invalid (DV = "0") approximately every 416 local clock cycles.

In the opposite case, the local clock may be 400 MHz and the incoming data stream 401 Mb/s. In this case, two bits of data will be generated on average every 416 clock cycles. In the case where the frequencies are nominally the same, for instance, two separate crystal oscillators, the separation in frequency will be a few ppm (in either direction), and it will be seen that circuit oscillates between producing zero, one, or two valid data bits per receiver clock, but the data is always correctly extracted.

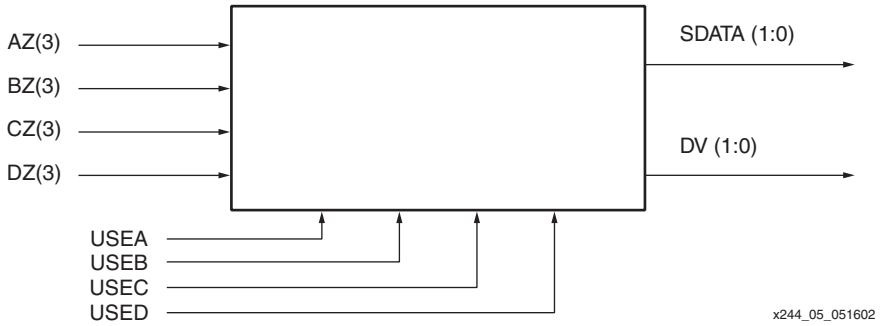

Figure 6: **Data Available Logic**

Using the Received Data

As discussed, on each receive clock cycle, there is either zero, one, or two bits of data generated by the data recovery circuit. In Figure 6 these are labelled SDATA(1:0) and their validity is indicated by DV(1:0). The first bit received is SDATA(0). SDATA(1), when valid, is the second bit received. These bits will normally be clocked into a FIFO for further processing either 8,16, or 32 bits wide. Assuming that the requirement is for 8-bit data, the logic required is a bit more complex than it first appears. Normally, the received data bits are passed into a shift register for "parallelizing," and when eight bits have been clocked in, the resulting 8-bit word is clocked into a FIFO. However, because the data recovery circuit can potentially generate two bits of data per clock, the case where the received bit counter goes from 7 to 9 needs to be considered.

The best way around this is to use a 9-bit shift register. If no bits of data are available in a clock cycle, the shifter and its associated counter remain static. If one bit is available, it is shifted in to the MSB, and the counter incremented by 1. If two bits are available, they are both clocked in by moving the shift register two positions and incrementing the counter by 2. If the counter reaches 8, then a valid byte is present in bit positions 1 to 8 of the shift register, and if the counter reaches 9, then a valid byte is present in bit positions 0 to 7 of the shift register. The valid data can, therefore, be passed to a FIFO (or the rest of the system directly) using a simple multiplexer.

Some example code for this circuitry is given in the design files as mentioned below.

It should also be noted that there is no guarantee that the first few bits received will be correct, as during this time the circuit will be "hunting" for the correct phase relationships. This applies to the first eight bits received, after which data will be valid.

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 3), 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.

Lock Requirements

The circuit has a lock requirement for a data transition to occur often enough to maintain data integrity. At least one data transition is required in the time that the circuit takes to drift one-quarter clock period. In the example in Figure 1, the data is arriving at 400 Mb/s, making one data period approximately 2.500 ns. One-quarter period is then 0.625 ns. The local oscillator frequency is 401 MHz; this equates to a period of 2.506 ns, making it (2.506 – 2.500) = 0.006 ns faster than the incoming data. The quarter period (0.625 ns) divided by 0.006 ns is approximately 100. Therefore, the circuit requires at least one negative transition every 100 clock cycles to function correctly.

By reducing the local oscillator frequency to 400.5 MHz, the clock cycle requirement number is increased to 200. If the received data is coded in some method such as 8b/10b, this will not be a problem, because an adequate number of transitions will exist. Care should be taken if the received data is a raw bitstream, because an adequate number of transitions may not exist. The design has been successfully tested at 420 Mb/s using a 23-bit Pseudo Random Binary Sequence (prbs) data pattern on the LVDS demo board that is using two separate oscillators for transmission and reception.

Simulation

By definition, the circuit uses two asynchronous clock domains. Simulation provides setup time violations and possible "X" propagation. This "X" propagation can be removed in MTI by using the "vsim+no_notifier" command. The "X" propagation can be safely turned off due to the arguments used in the metastability section.

Reference Design Files

The reference design circuit is implemented in HDL. It is fully synthesizable. The reference design files **(xapp224.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.

Conclusion

Virtex devices can be used to extract data from a serial link at speeds up to 160 Mb/s in a Virtex-E or a Spartan-IIE device, up to 320 Mb/s in a Spartan-3 device, and up to 420 Mb/s in a Virtex-II or a Virtex-II Pro device.

Revision History

The following table shows the revision history for this document

Date	Version	Revision
09/18/00	1.0	Initial Xilinx release.
01/10/01	1.1	Updated for Virtex-II series of FPGAs.
01/31/01	1.2	Edited Figures 1, 2, and 3, and added a Simulation section.
01/30/02	2.0	Circuit was redesigned to allow for slower and faster local oscillators.
03/28/02	2.1	Figure 5 updated to include both positive and negative transitions.
08/07/02	2.2	Clarified Figure 6.
03/04/04	2.3	Added references to Spartan-IIE and Spartan-3 devices. Added Figure 2.
01/24/05	2.4	Corrected Figure 5 and text in the paragraph underneath it.
07/11/05	2.5	Updated the data rate speed for a Spartan-3 device in the **Summary** section.