High-Performance Source-Synchronous Interfaces Made Easy

Webcast Date March 29, 2005

Presented by:
Sean Koontz, Applications Engineering Manager

Peter Alfke, Director, Applications Engineering

Slide 1:
Hello. I would like to welcome everyone from around the world. Thank you for joining us today for this webcast on "High Performance Source-Synchronous Interfaces Made Easy," brought to you by Xilinx and TechOnLine webcasts. Your presenters today are: Peter [Alfie], Director of Applications Engineering for Xilinx; and Sean [Koontz], Applications Engineering Manager for Xilinx. This webcast ultimately allows you to sit back and have the navigation ***. As the user participating in the webcasts, you will able to ask questions at any time during the presentation, by clicking on the "Ask a Question" button, typing a question in the top of the window, and clicking, "Submit." The presenters will be answering questions at the end of the webcast, but please ask answer them at any time. Also included with this webcast is a survey. Please take the time to open, fill out and submit the presentation survey. You can access the survey at any time in the "Print Documents" in the "Viewings" pull down menu on the left-hand side of your interface. This survey will also path open when you choose to close your viewer window, or when the viewer window closes automatically at the end of the webcast. By submitting this survey, you will be provided Xilinx and TechOnLine with valuable feedback on the subjects covered in the webcast, and also how we can improve the webcast product. And now it gives me great pleasure to introduce you to Peter Alfie.

PETER ALFIE:
Slide 2:
Hi. Good afternoon, ladies and gentlemen. Good evening in Europe and good early morning in Asia. This is the first of our webcasts. We covered performance, signal integrity, hour construction, and then the interface, as before. Today we will address source synchronous I/O.

Slides 3+4:

First, I will give some background. Then Sean Koontz will give an overview of the timing problems and their solutions ***. And he will follow this with detailed applications examples. This is again an engineering presentation with few, if any, retro *** research and stratics *** during that era. We really do not like their mudslinging marketing presentations. And we know that green *** engineers don't believe that kind of nonsense either. There are plenty of interesting verdicts or features to talk about in a positive way. Why source synchronous I/O? What's wrong with go *** system synchronous I/O? One central clock observes the whole board. Well, we are caught in a conflict between two giants.

Slide 5:
Gordon Moore [Ritache] postulated Moore's Law: the doubling of *** account every other year. As a side effect, system speed is doubling every five years, and it has done that vehemently. 40 years ago, we were struggling with one megahertz clocking. It's first hard to be the ***, ***. 40 years and eight doublings later, we are now at 250 megahertz bus copying, heading towards 500 megahertz, and one gigahertz every two years. It's really great, ever-increasing performance. But then there is Albert Einstein, who incidentally exactly how many years ago, postulated that the propagation of speed of electric signals can never be increased beyond the speed of light. In fact, the velocity on a PC board is only running half the speed of light, and therefore it is constant. That was no problem at one megahertz. We assume and allocate 25% of the clock area for the Interconnect Delay, and the rest for the BIOS delay, set-up times and margins. We, in those day, had 30 meters, 100 feet, available for the wiring. We put around wires all over the office or the home. Circuits were slow but interconnect was no problem at all. 35 years later -- that's five years ago -- at, let's say, 100 megahertz, we still put it for 30 centimeters, or 12 inches, of *** from the PC board. But today, at *** speed, we are down to six inches, and we are heading towards three inches. Obviously, we can not constantly build complex systems within a three-inch cube.

Slide 6:
Here is a different view of the same problem using circuit time: Years ago, good old system synchronous designed things that looped up terminator to drive 50 megahertz through devices, and each had about 10 nanosecond delay. No problem at all, even with two nanoseconds of *** delay. At 100 megahertz, and with five-nanosecond device delays, the timing got a bit tight. It was even starting using clicks, fly, centralized clocks, global

clocks and so on. And at 200 megahertz, this scheme doesn't work at all. Five nanoseconds not, period. Two nanosecond clock is solution. Two nanoseconds actually delay, one nanosecond interconnect delay brings absolutely nothing as in timing modules. That nice old idea of one common clock driving multiple clicks simultaneously has become a form literally of the past, purely extravagant. The only way out of this dilemma is source synchronous clocking, which means: We set the clock to *** data. This makes any interconnect delay irrelevant. The data on the bus arrives whenever it arrives, but it brings internal clock. This has some drawbacks: there are more clocks and more clock pins, and there even be multiple clocks, clock phases and checks. But we have more better made packages with many connections, and we have chip-internal *** to take care of that. There is also a detailed filing agent. The clock is cloven together with data. And the clock edge, therefore, arrives exactly at the time when it interchanges. And that is precisely the wrong moment the clock to behave. The good portion of the day, Sean and I will describe methods. We may handle the clock or the data, so that the clock hits the center of the data ***. That's referred to as clock ***.

Slide 7:
Here is ability of source synchronous interfaces. 10 years ago, the 66-megahertz clocks had achieved a data *** of a few gigahertz a second. Today, the RapidIO, HyperTransport in SFI-4, to give us well over 10 gigabytes for second data ***.

Slide 8:
So, synchronous interfaces have become the norm. They are used for point-to-point traffic, no *** rates or buses. They increase the chip-to-chip speed with 500-700 megahertz clocks. And they give better timing monitoring, and that means higher reliability. Some examples are listed here from the world of networking and data com, and also memory interfaces.

Slide 9:
Finally, I might also mention that the understood reliable center of the data is shrinking fast, even faster than the bit ***. Since the transitionings use up a disproportionate part of the bit period -- another reason we use the available timing budget very carefully. So, let me conclude: Source synchronous blocking is not just the better way. It's the only way in today's and tomorrow's ***. *** Sean Koontz to tell you all the details.

SEAN [KOONTZ]:

Slide 11:
Thanks for the wonderful free amphitheatre. I am going to take over now with what we, Xilinx and our customers have identified as some of the major challenges in designing with today's chips, source synchronous ***.

Slide 12:
I'll start with the first plot diagram of Vertex-4 I/O and some of the innovations that have been added to this architecture, in particular to deal with: with clock forwarding schemes as ***, finally called ChipSync. Every I/O includes these major features: ISERDES, OSERDES; those are our primitive names for the serializer and deserializer included in the I/O. The ISERDES block includes elements to divide the incoming clock frequency and distribute multiple versions -- or synchronous versions of clock. Also included are mechanisms for bit and *** alignment I'll talk about shortly. They also include our clock distribution networks in this block that he called ChipSync, it's the complete packaging -- the solution that we tile throughout the architecture in order to not only implement a single high speed source synchronous bus, but in order to have many source synchronous buses.

Slide 13:
The challenge number one: Data Capture at high speeds. This is fundamentally the most difficult in sequencing aspect of implementing high speed bus -- not only capturing data, but doing so and understanding which margins you have left in your system so that you can proportion a certain amount of error or slack to your PDP design or to your ASIC *** or whatever device you're trying to hook up to an FPGA. And obviously when you get up to a gigabyte per second data, you're electing a very, very small target to hit, which is what we're showing here. Objective simply to put that clock right into the middle of that available data window. So how do we deal with that?

Slide 14:
We manipulate the data or clock or both in order to center that clock edge in the available data valid window target. Every Virtex-4 I/O includes what we call IDELAY, under the name or a variable, have fine resolution, a derailment, which is fully user accessible. See this 64 tap delay chain with 75 per seconds resolution. And it is calibrated by a module called IDELAY control block. The provision or the user requirement is that a 200 megahertz clock is provided to that, either *** controlled block. We specify what the quality of that clock it could be in order to guarantee the 75 pica-second in our data

sheet. This allows us to delay data and audit in fine resolution increments while holding the clock still. This achieves obviously a fine sampling of a single data var, but also allows us to manipulate each data channel independently. So, I'm alluding to some information I'm going to get to later. But that's a key distinction. Objective number one is sampling data with the most precision. Objective number two is sampling the entire bus with the most precision.

Slide 15:
Challenge Number Two: Managing Clock Speeds Up to 700 Megahertz. This is not only a challenge to the FPGA or an ASIC designer, but it's a challenge to the PCB designer as well. We identified this early on with Vertex-2 and implementing high speed for a particular spot to *** it. You know, what are customers needed to do? In other words, what kind of clock qualities are being provided to our chips? What kind of reliability do we need to understand or limit that either source synchronous clock or clock source? What this all boils down to is: How do you distribute ISD *** clock inside its out-rig and maintain the highest precision duty cycle, and minimize the skew as you distribute the clock? With Vertex-4, we implemented all of our global talk networks fully differentially, with differential drivers and interconnects. This allows for higher speed and less duty cycle distortion. In addition to the global clock networks, we also added four fully differential IO clocks per bank; and I'll elaborate on those and what IO clocks are in a little bit. Now, the ability to forward clocks is also an interesting application of our architecture in that distributing high speed clocks on a board on a system is also challenging, getting require additional components. The FPGA can serve as a precision-aligned clock distributor as well, using the IO clock or the global clock ***.

Slide 16:
So, challenge number three: PCB Layout is becoming more and more of a challenge as these data rates increase. Obviously, with a compact board, layout constraints can result in trace length differences. Or, it can even *** cycle precision scheme matching in a highly compact board to become almost impossible. Propagation delays for connectors may also not be available; so you may not know, or you may have to tolerate a certain amount of skew that the connector vendor specifies. So these are unknowns that are attributed to the PCB layout itself.

Slide 17:
So, I mentioned the ability to bit to you skew individual

channels using IDELAY, right in the IOB free scaling. So, that allows you to align or to optimize the viewing quantity of each data. But if you have to skew in a bus, your words may be misaligned as well.

Slides 18 + 19:
We have a second block that we call Bitslip in every IOB, which re-orders the data stream coming into the ISERDES, and allows you to shift, basically, barrel shift that thing coming through, and find every possible combination of it -- this assumes a training pattern -- of the data stream coming into the IOB. This is also available in every I/O.
So, main objective is to word align. So we use IDELAY to optimize this re-uplink point for each IOD, and then we use Bitslip to make sure that the words are aligned as they go into the fabric.

Slide 20:
Challenge Number Four is implementing multiple interfaces. Multiple interfaces are going to require multiple unique clock domains. This makes clock management particularly pivotal and resource intensive; it requires synthesis, *** distribution issues. IO Placement for multiple interfaces is also a problem. You need to break out of the ball grid array, and you need a floor plan here for your PCB.

Slide 21:
So, I mentioned that the four IO clocks per bank. What is shown here: each four is on a blue line is a clock region. A bank consists of two clock regions. So, what's showing here is one and a half banks. The IO clocks are also paired up with what we call BUFARS or regional clocks. The regional clock divides down the incoming clock and then distributes it to the fabric and the IOs as well. This is how we handle the serial side or the parallel side of the ASERDES. There are 8 24-clock regions per device up to four clock-capable *** bank, as I mentioned before. The regional clock buffer drives fabric. It can be accessed by fabric. It can accessed by the IO clock as well.

Slide 22:
I've sort of boiled down the amount of clock resources that we have put into Vertex-4: 32 fully differential global clock inputs, as well as single-ended clock inputs. That means that we are not burning up half of our single-ended clocks in order to implement differential clocks. You get 32 single-ended or differential. As I mentioned before, 8-24 clock regions for device, 20 DCMs, 8 PNGDs -- for those of you who don't what the

PNGDs are, either phase *** that we got for Vertex-4, which is a new feature.

Slide 23:
We have also changed, or moved away from our IO ring methodology. So, with Vertex-4, we have introduced the *** architecture. This has an understanding for package design in that it provides a more symmetrical escape pattern, for all of our IOs. You can see by the way our banks are laid out in the architecture. This helps us now unskew both in terms of clock distribution network and in terms of package design. But how does this translate to assisting the PSB design? It provided even more flexibility in that you can implement more low skew source synchronous buses in more different locations, and it cross-ranks.

Slide 24:
This is just to summarize some of the resources in a different way. *** 240-960 *** IOs or SelectIO pins and up to 68 clock-capable I/Os, that's up to 68 different I/O ***, or 68 different unique source synchronous buses in all respects.

Slider 25:
Now our better offer is 45 degree steps or ***, and we're offering 75 picoseconds of resolution. As you increase in frequency, you can see where the tradeoff becomes quite significant. It's sort of how much actual valid data you really have? You have all kinds of results if I count your supply.

Slide 27A:
Let me move into discussing a couple of our applications that are key showcases for the source synchronous architecture, SFI-4 and SPI-4.2. This is a classic illustration of how our chips were being used in systems and where the value adequately comes into play. We have seen that multiple source synchronous buses and to have VGA is almost a requirement. It is very seldom that you see somebody *** use just a single *** point.

Slide 27B:
Chip function. What is shown here is a SFI-4 to SPI-4.2 bridge between a general processor and a OC-192 Framer. We also might have a SPI-4.2 to Serial for high upgrades in the backend of this part.

Slide 27C:
A classic example: four-way switches are also critical. So what is this really getting at? It's getting at the need to be able

to implement not only multiple buses, but to be able to move them freely around and have a lot of free flexibility in order to get these carts built.

Slide 28:
So, we've simplified the SFI-4 design with a lot of architecture changes of Virtex-4. We're leveraging ChipSync fully, moved these higher serialization/deserialization circuitry into the IOB, as well as the clock distribution. And the FIFO16 is a block for clock domain change, data handoff between the source synchronous bus to the core.

Slide 29
So here's a block diagram for the SFI-4 block. A trip to left side is the user interface: a 64 bit data interface, and the global clock. We hand that data to a FIFO. We time that; there's a transmit clock on a -- And then serialize it four-to-one, and then transmit the 16 bit bus with a forwarded clock using the output DVR registered to forward it, and you just change the line to data. And the receiver is the mirror of that process.

Slide 30:
Blocks used for the receiver: The recovered clock in its network are implemented on a BUFIO and a BUFR. The BUFIO is the High Speed Clock distribution network that I talked about earlier, the serial-side of the receiver. And BUFR is a divide-by-four version of the input clock, and that clock unit parallel-side of the deserializer and the fabric. Recovered data is recovered in the ISERDES. We have a block called ISERDES_ALIGNMENT_PROCESS, which I'll talk about shortly. This is a clock to data training algorithm state machine. And then lastly, a FIFO16 that's using the interface *** core ***.

Slide 31:
So, another block diagram showing the SFI-4 Receiver. Pretty much everything showing here is implemented in the IO. The ISERDES connected to Data 0-15. A Clock Capable I/O is used for the Reship Recover Clock. These are dedicated I/Os. These are dedicated locations in each bank. The dedication is really the connectivity to the BUFIO. The BUFIO is located at the end of every eighth clock row which is a clock region. And it sends out from there to the BUFR clock network and to the I/O clock network.

Slide 32:
So, as I mentioned before, we are doing a four-to-one and one-

to-four serialization/deserialization process. The *** design of Virtex-4 has been fair for us up to 700 Megahertz single-data rate, transmit at this point and receive. We do an automated version of clock data alignment, which we'll talk about in a minute. This is what we call bus alignment, and it does not require a training pattern. This design can also be used for XSBI and other high-speed single-data-rate LVDS applications.

Slide 33:
So, the ISERDES_ALIGNMENT_PROCESS. This is the bus alignment that I just mentioned. The objective is to align clock and data using IDELAY on each data lane. And this is data-agnostic, non-destructive training technique. The assumptions here are that clock and data are edge-aligned at the pins of the FPGA, and that the clock will be toggling at startup for several milliseconds before data is sent; in other words, that there will be some period of time that the clock will be switching at the input of the FPGA. The reason for this is that we are going to train to that clock. The clock is at now in 1,0 pattern. When we find the center of the sampling window for the ISERDES in the clock IOB and we move data to an optimal possible location based on that clock siting pattern. This fully has been implemented and fully characterized.

Slide 34:
So here's a circuit diagram of the clock training circuit. Now this can run all the time. As I mentioned, it's non-destructive. The clock is routed through the ISERDES in two different ways: one, it's passed through combinatorially and then *** the BUFIO and the BUFR. From there, those clocks are fanned to the data lanes. That is actually is what is physically recovering data. We also route the clock to the IOA circuit into the flip-flop in the ISERDES. This is a parallel path. From there we take a look at the output queue of this flip-flop in our state machine, and manipulate the delay in order to interrogate the clock eye.

Slide 35:
So, the best simplification of the algorithm that I could do here: We begin incrementing the delay on clock until we see a 1 to 0 change at Q output. Then we start counting the number of taps as we continue to implementing on clock and so we say a 1/0 change in the Q output. Then we start counting the number of taps as we continue to increment, looking for the next transition, the next change in state. Once we see that next change of state, the tap count equals the data valid window width of the clock at that register. And if you subtract that

final tap-delay value by half the number of taps determined to equal the data valid window width, you will find the absolute delay required to put the clock in the center of the data eye. Once you have that number, you increment all data channels by that amount and the data to clock alignment is complete. As I mentioned, it can constantly be running. So, if you have a voltage and temperature shift, you will catch that if it's greater than 75 pica-seconds, you will catch that by monitoring the clock count. And you can go ahead and move data once you have seen that. If the shift is such that -- If it is sudden enough that there is now way you can recover, you may lose data. But that is an extreme case, and there is probably going to be more problems in the system than the launch of the data on this bus.

Slide 36:
The transmitter in Virtex-4 is very straightforward. We use BUFIO and BUFR for the clock distribution, very similar to the receiver. We use the OSERDES block to serialize the data and the FIFO16 to move data from core to interface.

Slide 37:
Not surprisingly, the block diagram looks very, very similar to the receiver; everything is very symmetrical and very packed. It's all packed into the IOD. The first thing I would like to point out, and this is true for the receiver block diagram: What's shown here is three clock regions. It does not take three clock regions to implement this interface. This is how spread it could be. This is spread across one and a half tanks ***. But you can connect this entire bus in a single clock region.

Slide 38:
Now we are going to move on to this SPI-4.2 up to core. Before I do, everything I just mentioned about SFI-4, this is available as a reference design, in "Application Help." I'll have a URL at the end of the presentation. The SPI 4.2 information I'm about to share is based on our ID-coristine *** development work, and this is available at the core.

Slide 39:
So, Xilinx SPI-4.2 in Vertex 4 is fully compliant with OIF-SPI4-02.1. Ideal solution for packet over sign of ATM, and Ethernet applications. Supports OC-192 line speeds 10 gig and beyond. Supports both static and dynamic alignment modes that are laid out in that delay specification. It is a point-to-point interface and fully symmetrical. 16-bit data bus is LVDS.

Indeed we are IO. And it is supported in all projects or device.

Slide 40:
Now, DPA or Dynamic Phase Alignments is a version of bit skew which I alluded to previously. So, the objective is obviously to center sample data. But also, it's toward align. So, the SPI-4.24 uses both uses both IDELAY and Bitslip in order to compensate for channel skew. And it is capable and characterized to compensate for up to plus or minus one bit period skew.

Slide 41:
Advantages of DPA. Independent sample point determination for each bit is a key advantage. That means that we are identifying the most optimal sample point within our data valid window. And this leaves the most amount of system timing margin on the table for the rest of ***. This supports, is required for data rates over 700 megabytes, 800 megabytes. I don't know exactly where the SPI-4.2 cutoff point is. And we have the reduced the size of this core by 50% with respect to Virtex-2 and the Virtex-2 Pro implementation.

Slide 42:
So, on the black diagram, you can see the user interface, which is a 64-bit interface. The SPI-4.2 implementation is also four-to-one to elevation, 24 deserialization. They use a data source from Tree FIFO for the core to user logic company and status memory where all of the flags on the interface. *** resources used. It's not.

Slide 43:
So, the SPI-4.2 metered components in DPA, very similar to what I was mentioning with SFI-4. Using I-34 receiving, an IDELAY chain, Bitslip module, and the chip ***. Data recovery using the IDELAY state machine which is similar to the platform *** methodology. However, it is making use of the SPI-4.2 training pattern to interrogate actually each data bit independently. So, each data eye is interrogated and optimally set for the SPI production training pattern at start-up. And then lastly, bus de-skew algorithm uses Bitslip to reorder the output of each eye series such that each channel is word aligned going into the fabric.

Slide 44:
So, quick summary encore. A gigabit per second for pin data rates are available. We have reduce FPGA resources by 35%. I

mentioned 50% before; I think that is with respect to the entire core and IO resources, not just the fabric. In/outs are flexible for this core. We've reduced power in the Vertex-4 architecture. *** greater than four cores in a single device. And our data capture mechanism is even more accurate ***.

Slide 45:
This is the ML450 development board that my team designed. This is used for verification of those designs that I just mentioned, the SFI-4 implementation, and the SPI-4.2 implementation. Both the core and the application can be implemented on this board and checked out. We use Fantag *** and differential precision interconnect cable for loop-back, so we can talk for ourselves basically. You don't have to go board-to-board. We have our customers use this board and others like it to talk to their own systems by building paddlecard *** adapters, *** cable connectors to their own card. This is available today. It can be ordered. I'll show the URL at the end of the presentation. Or you can contact direct local FAE and get a demo or possibly borrow the board.

Slide 47:
And to sum up here, the key challenges that we've identified and solved with this architecture: Data capture at high speeds. Clock-to-data centering at "run time." There are multiple ways to do this. We have numerous application notes that go into detail on each different method, and they'll answer a lot of questions I'm sure I've left open; Managing clock speeds up to 700 megahertz. We have quadrupled our clock resources and made them fully differential; PCB layout challenge. IDELAY and Bitslip to eliminate *** skew is critical. Our in-package design and chip floor plan also helps in providing more flexibility in terms of what items you need to choose, from the FPGA designer, and from the PCB designer's perspectives. That also address the last challenges, implementing multiple interfaces.

Slide 48:
So, here's the URL I mentioned. Leverage the complete hardware verified solutions to assure first time design success. You can access this link to find information and contact on the core, SPI-4.2 or RapidIO. Also, the application notes for SFI-4, or a single-data rate 16-bit LVDS is more accurate; also information on the ML450 source synchronous interfaces toolkit. You have the reference designs, schematics, Gerber files, everything you need to get going. OK, at this time, I'd like to open it up for questions. It looks like we have a few here.

Question and Answer Session

Q: So, our first question, "What is the maximum IO speed of Vertex for general purpose IO?"

A: LDS IO maximum speed is 98 per second EDR. And single-ended IO including different *** to scale is I believe 300 -- 500 megahertz per second. No, I'm sorry, that's 300 megahertz per second.

Q: Are the data clocks *** to prevent reflection?

A: I'm assuming, you mean, the recovered clocks, are they terminated on the EGA? Yes, for LVDS inputs, all LVF inputs, we have on-ship *** determination natchpees *** that you set ***.

Q: Is data centering using *** block performed continuously?

A: Yes, this can be done continuously. You have full access to increment and accruement *** that perhaps as a lay element. And reset, so that is all done in the fabric. True, what really is meant by continuously? There is one implementation that we call window monitoring. We use the fairwell *** side of the input tree. I'll draw you a picture. Every IOD has a master and slave side. For any LVDS input, you will use the master side IO logic and not the slave side IO logic; that's redundant. So, what we do is route all data channels to the slave side and clock that into the fabric, so we're creating a mirror of all the data streams coming in. By doing that, we can manipulate the mirrored version of the data all we want without destroying the incoming data. So we can continually interrogate the data eyes, looking for differences between the mirror and the real data. And if we see that the real data has slipped with respect to the mirror, we can move the real data without destroying it. That's the most optimal way of insuring the best sampling over voltage and temperature and time.

Q: Is this source synchronous implementation process automated via synthesis, or does the user manually instantiate elements such as Bitslip?

A: Bitslip will appear in the pin in the primitive view. So, the user will manually instantiate an ISERDES primitive, and they'll have access to pins on their primitives, like the Bitslip enable pin and the IDELAY increment/accruement pins. So, and those are all part of one big primitive. The process is not entirely automated. And most of it is implemented in hardware, so you're going to have to substantiate primitives. It's only a handful of primitives that we're talking about. We're talking a BUFIO, a BUFR and ISERDES to build a receiver, basically. The catch is,

in order to implement word aligning algorithms and data centering algorithms, you need to use the fabric; there needs to be some intelligence from a state machine. So, the answer to that question is: Well, yes and no.

Q: Does Vertex core have support for SFI-5?

A: Not at this time.

Q: Did you have to resolve word alignment with the DDR-2 interface? If yes, what did you use?

A: Yes. You do have to resolve word alignment issues with any interface that has *** skew. So, if you're doing per bit deskews, or deskewing each lane, you have to address the word alignment issue ***, even if you know the word alignment interfaces are perfectly aligned. You need to know in the fabric that your words are in the right place. So, we use this procedure to do that, which is fully documented. You can take a look at the source code on our website, for -- I can't remember the application note number, but look for the DDR Vertex for I/O application note.

Q: What is a typical jitter using DCMs in Vertex-4? Do you recommend using Oxitclot *** generation for ***?

A: This entirely depends on how fast your interfacing needs to go. Typical jitter: The jitter is specified for RDCMs. I can't remember off the top of my head what that stack is, but you should be able to get that off the website. This is entirely up to you. I can't answer that because I don't know what you're talking to and how much *** you need in the system. We did implement a bus using DCMs and NLCore-50 ***, and we're able to transmit over about 18 inches of cable and two connectors *** receiver, and we're able to recover data with *** all the way up to -- I gave you the per second *** DCM per transmitter. So, that's a tough one.

Q: Is I-the-Way *** available in Vertex-2 Pro also?

A: No it is not available in Vertex-2 Pro. I-the_way is new to Vertex-4.

Q: How do you determine the sampling point when a training pattern is not available?

A: That's a great question. So, with SFI-4, what we showed was a method for training to the clock. The sample of the clock because we know what the pattern is. You know, it's one *** pattern, so we're able to find the edges of the clock, and thus we're able to determine what the center of the data eye is for the clock. The assumptions for training to the clock are that the bus has minimal skew and ***. You get a very similar thing for *** in the ***, and unfortunately I'm not an expert in *** but you can get the

application off the web as well.

Q: What do you use for a training pattern in the DDR-2 interface?

A: VDR-2 -- If you're talking about VDR-2 memories, don't -- I don't believe that there is a training pattern for the VDR-2 memory. For the SPI-4.2 interface, we use a *** of the ten ones pattern, which is specified in the ***.

OK. If there's no more questions, I'll hand it back to the operator.

Operator:
Thank you very much, Sean and Peter, for your presentation. And I would like to thank everyone for attending today's presentation of "High Performance Source Synchronous Interfaces made Easy," brought to you today by Xilinx and TechOnLine webcasts. I would like to remind you to please fill out and submit the survey. This survey will open when you choose to close your viewer window, or when the viewer window closes automatically at the end of the webcast. By submitting this survey, you will be providing Xilinx and TechOnLine with valuable feedback on the subjects covering the webcasts, and also we can improve the webcasts further. This presentation will be available to all *** users in on demand format. You will receive an e-mail with information how you can access the on demand version of this webcast. Thank you again for attending. We hope you can join us for future on-line webcasts. For a current schedule and on demand events, please go to www.techonline.com.

End of XilinxMar2905 PeterAlfie SeanKoontz Webcast

15