LatticeSC PURESPEED I/O Adaptive Input Logic User's Guide

June 2010 Technical Note TN1158

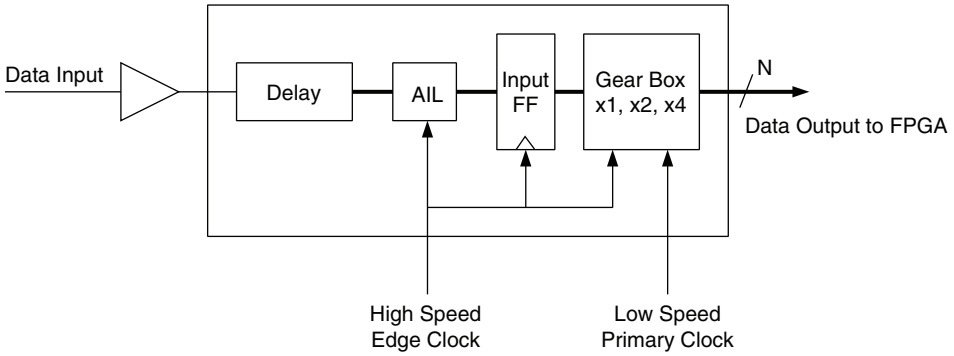
Introduction

Today's high speed synchronous interfaces pose challenges to the designer in maintaining clock-to-data relationships, managing data-to-data skew, and sustaining jitter tolerance. Many next-generation interconnects use SERDES based interfaces where the clock is embedded inside the data signal. SERDES-based interfaces, however, provide challenges in other areas such as data signal encoding, run length, and power. Lattice has developed an innovative technology to provide data recovery using virtually any input pin on the LatticeSC™ device. A typical CDR oversamples data by using multiple phases of a reference clock. The novel approach of the LatticeSC AIL oversamples the data using multiple phases of the data. This unique method uses less power and provides receiver performance that in many cases exceeds traditional CDRs.

LatticeSC PURESPEED™ Input Architecture

The LatticeSC PURESPEED™ Input Architecture provides a rich set of features for the designer of high speed interfaces. Figure 1 provides a block diagram of the major contents of the Input Logic block.

Figure 1. LatticeSC Input Logic Block Diagram



From the PURESPEED Input buffer, a signal at up to 2.0Gbps is routed to a programmable delay block. The delay block can have either a static, user programmable, or AIL monitored delay. After the delay block is the first stage input register. This register is typically clocked on the high speed edge clock. Edge clocks are dedicated high speed clocks that are arranged on the edges of the LatticeSC device intended to be used for clocking I/O interfaces. After the first stage register is the Gearbox which can divide down a high speed clock x1, x2, or x4 in either single data or double data rate modes. This allows for a 2.0Gbps data input to be divided down to an 8-bit bus operating at 250MHz in the FPGA. The transfer from the high speed edge clock to the low speed FPGA primary clock through the gearbox is guaranteed when using a on chip dedicated clock divider. Primary clocks are low skew global clocks on the LatticeSC device. Dedicated clock dividers can be used to divide down high speed edge clocks to lower speed primary clocks.

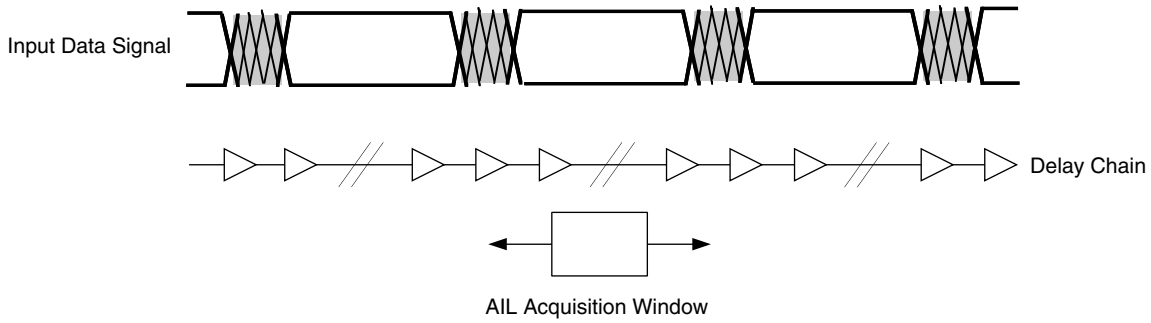
The LatticeSC AIL coupled with the Gearbox provides a complete high speed receiver for the LatticSC device.

Overview of the Adaptive Input Logic

The Adaptive Input Logic (AIL) provides the ability of the input logic to dynamically find a solution by monitoring multiple samples of the input data. The input data signal from the input buffer is run through a delay chain. Data, transitions, jitter and noise are all contained inside of the delay chain. The AIL will then search the delay chain for a clean sampling point for data. Once found the AIL will monitor and walk with the data dynamically. By continually looking at clean data samples the AIL can walk in both directions once a stable location has been found. This novel

approach of using a delay chain to create multiple copies of the data provides a lower power solution than oversampling data with a higher speed clock. Figure 2 provides a high level view of the AIL methodology.

Figure 2. LatticeSC AIL Delay of Input Data Waveform



The AIL slides the acquisition window through the delay chain searching for stable data based solely on data transitions (i.e., data with no transitions due to jitter, noise, etc.). A specific training pattern is not required to perform this bit alignment, but simply the presence of data transitions. The size of the acquisition window is user selectable allowing the AIL to operate over the full range of the PURESPEED I/O range. Based on dynamic user control the AIL can either continuously adjust the window location based on data edge detection or can be locked to a specific delay.

The AIL operates on single data and double data rate interfaces and is available on most FPGA input pins on the LatticeSC device and all buffer types. The AIL block is low power using only 0.003 mW/MHz typical (6 mW @ 2 Gbps) for PRBS 2^7 data. Multiple AIL inputs can be used to create a bus with a FPGA circuit to realign the bus to a common clock cycle.

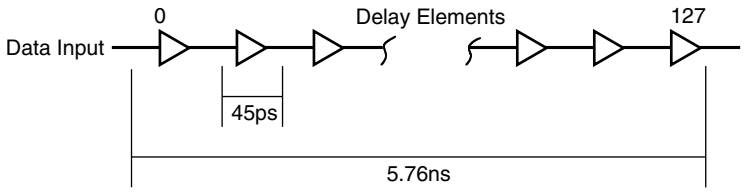
How the AIL Works

The AIL works on the principal of a sliding window. The AIL will move this sampling window into the data eye where no data transitions occur. This section describes the mechanics and search algorithm used to accomplish the alignment.

The Delay Chain

Inside the LatticeSC Input block is a delay chain. When the AIL is used this delay chain is used by the AIL to provide delayed copies of the input data signal. The delay chain, when using the AIL, can delay a data signal up to 5.76ns in 45ps nominal delays for a total of 128 delay settings or taps. Shown in Figure 3 is a diagram of the delay chain.

Figure 3. LatticeSC AIL Delay Chain



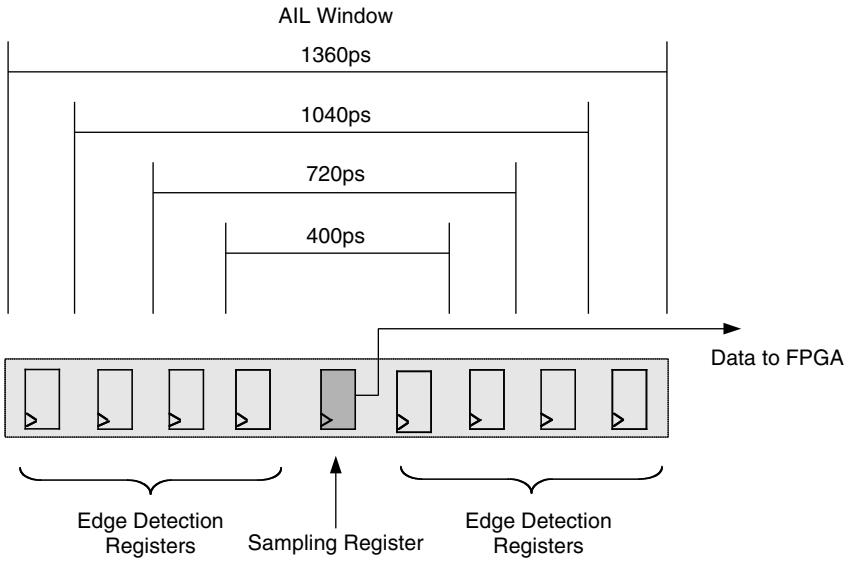
At any given time there are multiple consecutive periods of input data in the delay chain available for sampling. For example, with a data rate of 1.25Gbps the data period is 800ps providing over 7 full periods of data in the delay chain.

The AIL Window

The AIL window is used to take samples of data from the delay chain. The window contains edge detection registers and the actual center tap data sampling register. The middle register known as the sampling register is the

actual register where the data is registered and passed onto the Gearbox and FPGA. The larger the window selection the more edge detection registers are available up to a total of 4 edge detection registers on each side of the sampling register. Figure 4 shows a diagram of the register layout of the AIL window with the available sizes.

Figure 4. LatticeSC AIL Window Register Layout and Available Sizes



Between each register are two delay elements or 90ps of delay. Not shown in the figure is a second set of registers on the negative edge of the clock. This negative edge window is used along with the positive edge window when operating in a DDR mode.

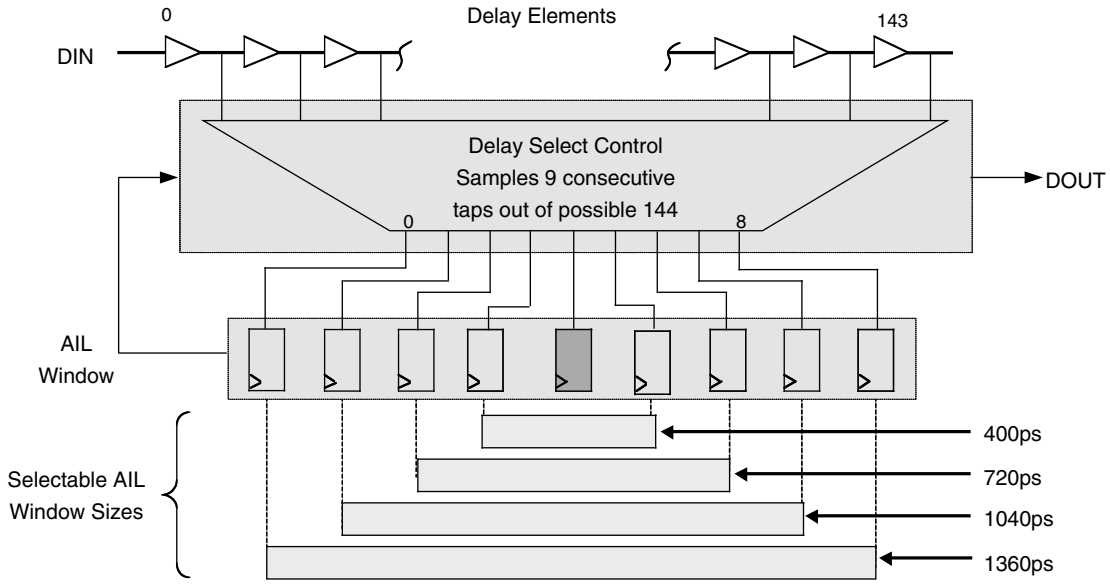
Note: The window size is labeled based on its Worst-Case Slow total delay. Total delay is a function of raw number of delay elements and setup/hold margins of the edge detection and sampling registers.

The AIL Search Algorithm

By default, the AIL window slides through the delay chain looking at different delayed copies of the input data present on each register. The AIL will use the edge detection registers to determine the location of data transitions. The AIL will move the window to a specific location of the delay chain where no data transitions are present to ensure that the sampling register captures correct data.

Figure 5 show how the delay chain is used in conjunction with the AIL window.

Figure 5. LatticeSC AIL Delay Chain and Window Diagram

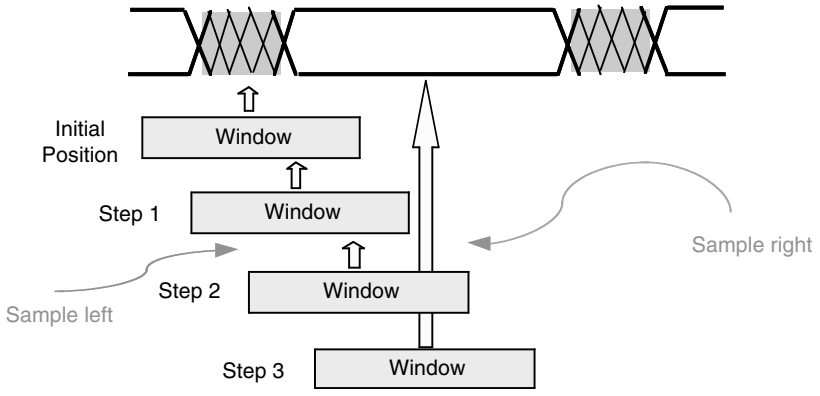


The AIL begins its search for a clean window position in the middle of the delay chain.

When four consecutive data transitions are detected inside and on the same side of the window the window will move one step (90ps) away from the detection. After the move the AIL will wait again for four data transitions on the same side of the window before moving away from the detection. In the unlikely event that the AIL continuously identifies data transitions on both sides of the window a proprietary algorithm is used to continue to move the AIL window away from the data edges which prevents lock up. Data transitions on each side of the window can most likely happen when a window size that is too large is selected. This topic will be discussed in a later section. If data transitions are not present, the AIL window will not move.

Figure 6 provides a visual representation of the search algorithm by way of an example.

Figure 6. Example of AIL Search Algorithm



The example in Figure 6 provides a worst case example of the AIL starting in the center of a data transition. The receive jitter of the data transition will determine the width of the data transition area the AIL will need to move away from. After identifying four data transitions on the same side of the window, the AIL will determine a move direction. Step 1 is the first move position which happened to be to the right in this example. After this step, the example shows that the window is still in a data transition rich environment. After four data transitions on the same side (the left side in the example) the AIL will once again move to the right to Step 2. Once in Step 2 the middle sampling register is clean and valid data can be seen by the user, but there are still data transitions on edge detection regis-

ters. Again after four data transitions the AIL shifts the window again to Step 3. At Step 3 the window is completely clear of data transitions. The window will remain in this location until a time when four data transitions are detected on the same side of the window.

Over time, a variety of system conditions may change resulting in AIL movement. These include temperature, voltage, data jitter, and clock jitter. Temperature and voltage impact FPGA routing delays for both the clock and the data. A change in either delay will result in data transitions moving with respect to the clock edge. If a delay change causes transitions to enter into the window then the AIL will compensate by adjusting the window as necessary.

Data jitter and clock jitter can also force the AIL to move the window. Jitter will be discussed in the next section.

A Note on Data Patterns

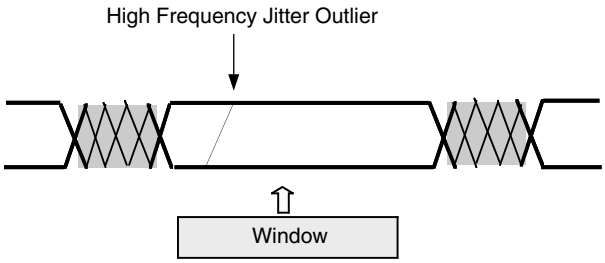
The AIL does not require a specific data pattern (or training pattern) to find a clean location. The AIL operates only on data transitions. The greater the density of transitions in the data pattern the faster the AIL can converge on a clean location for the window. If no data transitions occur then the AIL will not move the window location. Therefore the AIL will not drift in the absence of data transitions.

How the AIL Handles Jitter

There are two types of jitter, high frequency jitter and low frequency jitter. High frequency jitter is typically thought of as cycle-to-cycle deviations. This type of jitter is usually created by the implementation of the hardware. Low frequency jitter is sometimes referred to as wander. This type of jitter is measured in Hz-kHz and is typically created by the clock source. As a receiver, it is usually best to tolerate high frequency jitter and track low frequency jitter.

The algorithm and window design of the AIL allows the LatticeSC receiver to tolerate high frequency jitter and respond to low frequency jitter. High frequency jitter is tolerated by the AIL window's edge detection registers. Figure 7 shows a scenario of high frequency jitter.

Figure 7. AIL High Frequency Jitter Tolerance



Using the AIL, once the acquisition window has found a location free of data transitions the edge detection registers act as a buffer against high frequency jitter. Figure 7 shows a high frequency jitter outlier deep into the AIL window. If this transition occurs four times in succession, the window will move. This first single transition will be tolerated by the AIL because it is not deep enough to impact the sampling register in the middle of the window. Window size selection will directly impact the amount of additional high frequency jitter tolerance of the AIL. Table 1 provides a listing of AIL tolerance per window size. The AIL tolerance is only a portion of the total jitter tolerance measure.

Table 1. AIL Window Jitter Tolerance

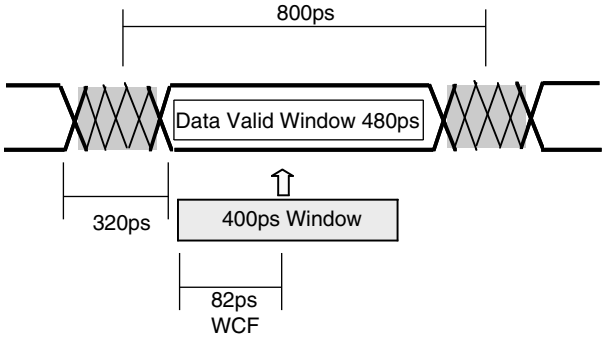
Window Size	WCF Jitter Tolerance	Typical Jitter Tolerance
400ps	82ps	105ps
720ps	147ps	189ps
1040ps	215ps	276ps
1360ps	284ps	369ps

High Frequency Jitter Tolerance Example

To express the high frequency jitter tolerance of the AIL, it must be calculated at a specific rate and under specific conditions.

For example, what is the jitter tolerance of the AIL at 1.25Gbps with 320ps of input data jitter? Figure 8 shows a model to perform the calculations.

Figure 8. AIL Jitter Tolerance Example

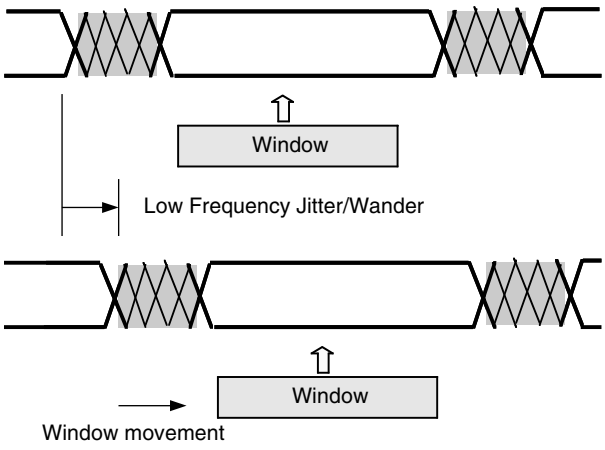


At 1.25Gbps the period is 800ps. To find the data valid window the jitter is subtracted from the period (800ps - 320ps = 480ps). With a data valid window of 480ps, the 400ps AIL window size is selected. When the AIL begins searching for a clean location, the window initially encounters jitter. Over time, the AIL will move to a stable location just outside of the jitter. For example, with 320ps of jitter on the data this stable location would be 320/2= 160ps from the expected data edge location. At this point the AIL tolerance is 160ps of movement + 82ps of AIL tolerance = 242ps. To convert this into a unit interval value simply take 242/800 = .30UI of jitter tolerance. This tolerance is specific to the window size and system jitter.

Low Frequency Jitter

The built-in feature of seeing four separate detections before moving the window allows the AIL to adjust slowly to low frequency jitter (or wander). With the hysteresis of algorithm for data transitions the AIL acts as a low pass filter for jitter. Figure 9 shows how the AIL responds to low frequency jitter by moving the window.

Figure 9. LatticeSC AIL Low Frequency Jitter Response

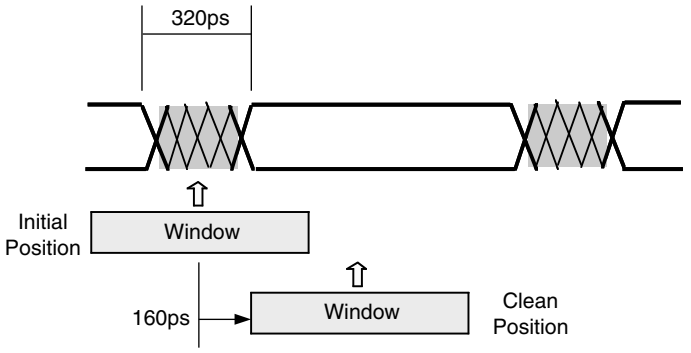


Acquisition Time

Receivers measure the time it takes from data present on the input to the sampling of correct data as the acquisition time. The acquisition time is very dependent on the architecture and implementation of the data recovery circuit. It can also be dependent on the amount of data transitions and jitter. The AIL acquisition time can be

calculated based on the total jitter of the system and the data transition density. The AIL requires four data transitions for the AIL to perform a move operation. The more transitions there are in the data pattern, the faster the AIL can move to a clean location. Again, an example will be used to show the calculations. In this example 320ps of total jitter will be used with the assumption that the jitter will be distributed on one side of the data window. Figure 10 shows a model for the calculations.

Figure 10. Model for LatticeSC AIL Acquisition Time Calculation



The worst case starting position for the AIL window is directly in the middle of the data jitter. Starting in the center of the jitter window will require the AIL to move 320ps/2 = 160ps to a clean position. Using typical delay numbers allows the AIL to move 90ps per location. This particular example requires two moves (90ps x 2 = 180ps) to move to a clean position. The AIL requires four data transitions inside the window to determine a move direction. Two moves requires 2x4=8 data transitions to find a clean location for the data window. Again, this example assumes a jitter distribution that is weighted to one side of the window. It is possible in some systems for the jitter window to straddle the data window. In this instance the window will take longer to move and directly impact lock time.

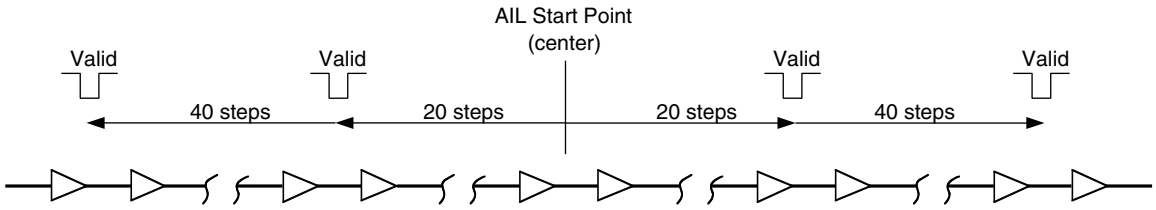
In reality the AIL will be sampling clean data prior to moving the window completely out of the jitter. This is due to the AIL window tolerance and the fact the center tap register is clean. This early clean sample time is not included in the calculations.

A Finite Delay Chain

Although the delay is relatively long, it is possible that the AIL could eventually shift to the end and exhaust the delay chain. If starting in the center of the delay chain with the smallest window size selected this would require a shift of approximately 3ns. It is unlikely that a shift of 3ns would be required to find a clean sampling location. However, in systems that require a frequency offset between the transmit data clock and the receiving AIL clock this condition could occur. When the end of the delay chain is reached by an edge of the window, the AIL will reset itself to the middle of the delay chain and restart the search algorithm. During this process the data may be sampled incorrectly until the AIL finds a new location for the window.

The Valid output of the AIL indicates delay chain movement to the user. Figure 11 shows a diagram of the behavior of Valid as AIL moves through the delay chain.

Figure 11. Valid Behavior



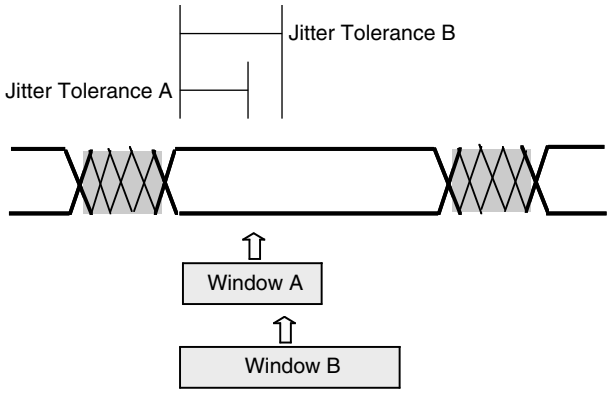
As the AIL slides through the delay chain, the Valid output will create a pulse that is four high speed clock cycles low to indicate movement to the user. This can be used as an indication that the AIL is moving through the delay chain and not stabilizing on a specific location.

Selecting a Window Size

As described previously, there are four sizes of AIL window from which the user must select. The user should select the largest size window possible that will fit into the data pulse width. The data pulse width should be considered as it enters the LatticeSC device, not the ideal data pulse width. Transmit jitter, board trace, power supply noise, and many other variables impact data jitter and must be taken into account when calculating the actual data pulse width.

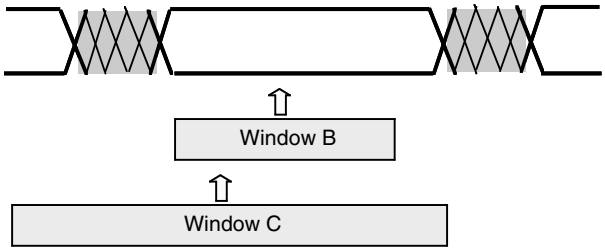
If a window is selected that is too small for the application it will result in less high frequency jitter tolerance. Figure 12 compares two windows sizes and their jitter tolerance.

Figure 12. Selecting Too Small of a Window Size



If a window is selected that is too large the AIL may never find a clean location. Figure 13 shows two window sizes and their locations.

Figure 13. Selecting Too Large of a Window Size



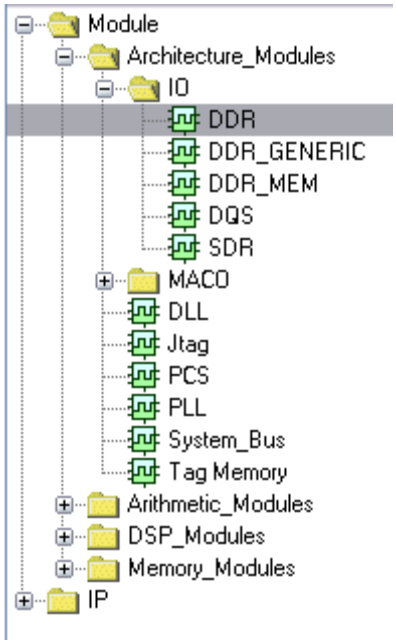
Assuming that the AIL started in the middle of the data transition on the left edge of both windows, the AIL would move to the right until no data transitions are detected. Window B quickly finds a solution. Window C keeps moving to the right until it reaches the end of the data period. Window C has now moved to the start of the next data period and begins finding data transitions on the right side of the window. At this point the AIL can no longer move to the right and will remain in this location since it sees edges on both sides of the window. However, as shown in the diagram, the sampling register is not at the ideal location and has less jitter tolerance compared to Window B.

The Lock output of the AIL is useful in determining if a window size is too large. The Lock output will go high 32 clock cycles after the window has stopped moving. If the Lock output never goes high then the AIL cannot find a clean location for the window and the window size selected may be too large. This does not mean that the sampled data is invalid, only that the window location is still moving. One item to note is that the Lock output will not go high if the AIL continues to walk the delay chain. In this case the Valid output can be used to monitor if the AIL is walking the delay chain.

Using the AIL

The AIL block itself is supported inside of the LatticeSC I/O Logic block. The user can create a LatticeSC I/O Logic block HDL module using the ispLEVER® tool IPexpress™. This module generator will create a customized HDL module which provides support for AIL. Figure 14 provides a screen shot of the available I/O modules in IPexpress. The LatticeSC AIL is supported using either the DDR or SDR modules.

Figure 14. IPexpress Module List Screen Shot



Shown in Figures 15 and 16 are two screen shots from IPexpress for the LatticeSC input logic block. The first is the Configuration tab and the second is the Advanced tab.

Figure 15. IPexpress LatticeSC Input Logic Block Screen Shot, Configuration Tab

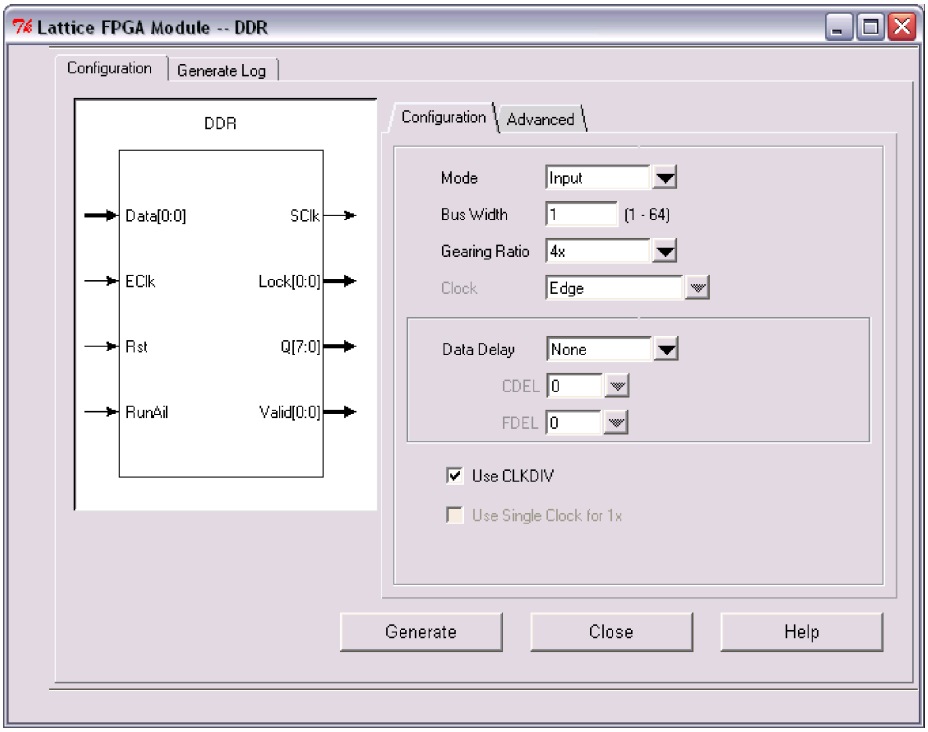
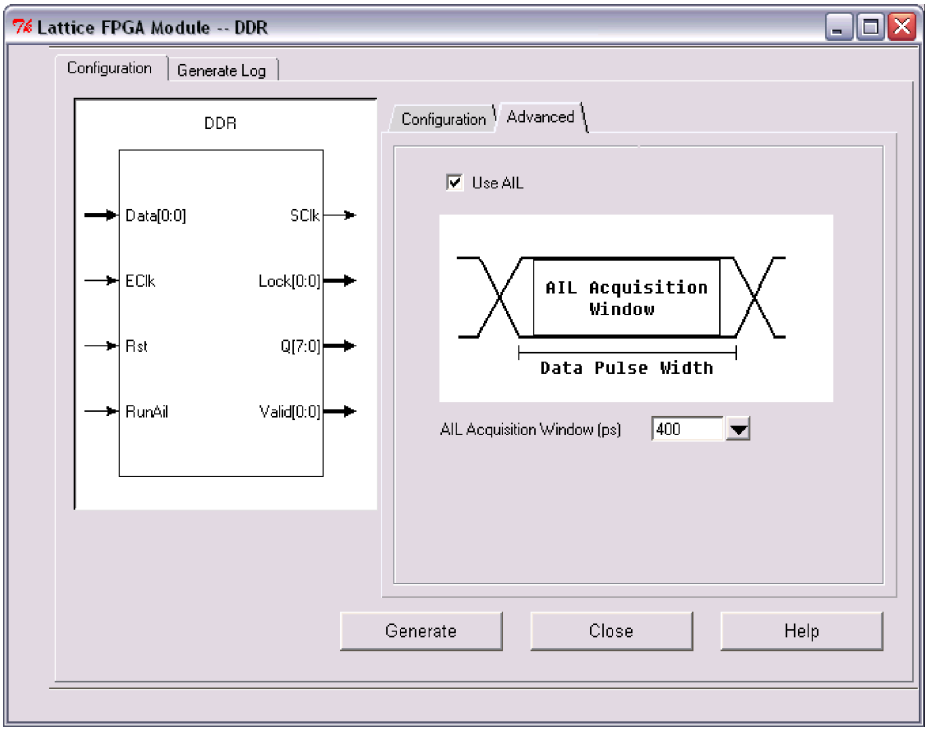


Figure 16. IPexpress LatticeSC Input Logic Block Screen Shot, Advanced Tab



In addition to AIL support, the LatticeSC I/O Logic block also provides a gear shift feature which allows the LatticeSC PURESPEED I/O to run at speeds up to 2Gbps while running the FPGA interface at only 250MHz. Each of the

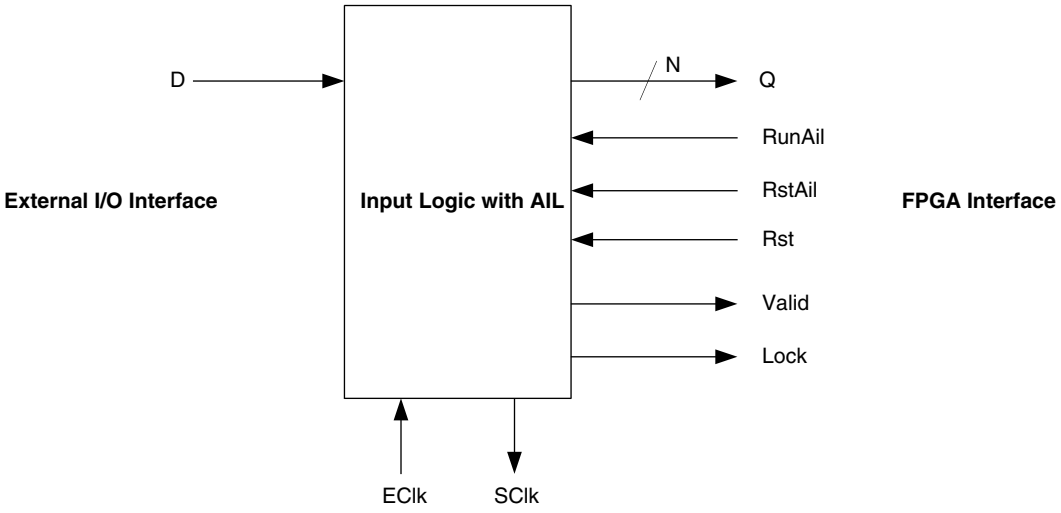
user options will now be discussed in the context of using the AIL. For more information on the full capability of the LatticeSC I/O Logic please refer to technical note TN1088, *LatticeSC PURESPEED I/O Usage Guide.*

- **Mode** - The selection should be set to Input when using the AIL
- **Bus Width** - This selection should be set to 1 when using the AIL. When creating a bus in IPexpress the data is striped across the gear shift bus. This is not what is desired when trying to align multiple AIL bits. In order to create a bus of AIL it is better to create a single bit and instantiate that module as often as required. Multiple AIL alignment is discussed later in this document.
- **Gearing Ratio** - Selects the gearing ratio of the I/O logic which an be x1, x2, or x4. In a SDR module the Q output bus will be either 1, 2, or 4-bits wide. In a DDR module the Q output bus will be either 2, 4, or 8-bits wide.
- **Clock** - This selection should be set to Edge when using the AIL.
- **Data Delay** - These options are not used when selecting AIL since AIL will be controlling all of the DELAY options for the user.
- **Use CLKDIV** - If selected, a LatticeSC CLKDIV will be inserted into the module to create the low speed clock for the user that matches the gearing ratio selected
- **Use AIL** - Select this checkbox to enable the AIL. This provides the ports for the AIL control on the module.
- **AIL Acquisition Window (ps)** - Selects the window size AIL will use to achieve a clean data eye. There are four windows sizes available which are 400ps, 720ps, 1040ps, and 1360ps. These window sizes are based on worst-case slow delays.

AIL Ports

The AIL ports are included with the LatticeSC Input Logic module. Figure 17 provides a block diagram of the LatticeSC Input Logic module created in IPexpress.

Figure 17. LatticeSC Input Logic with AIL



The I/O ports of the LatticeSC Input Module with AIL are listed below.

- **D** - This is the high speed data input from the buffer.
- **Eclk** - This is the high speed clock used to sample the high speed input Data. Eclk must come from an edge clock route in the LatticeSC.

- **Rst** - This async reset is synchronized internally via the CLKDIV to the Eclk clock and will reset the I/O Logic block gearing function.
- **RstAil** - This async reset will reset the AIL block to the center of the delay chain. The Lock output will go low during this reset.
- **RunAil** - This active high async signal will allow the AIL algorithm to adjust the location of the window. When low the AIL will not adjust the delays. When RunAil is asserted the AIL begins searching the delay chain from the point where RunAil went low. The output Lock will go low each time RunAil is first asserted.
- **Sclk** - This is the low speed geared down output clock sourced from the CLKDIV inside the module. This clock is routed on a primary clock.
- **Lock** - This async signal is used to identify if the window size selected fits inside the data period. Lock does not have to go high for the AIL to sample correct data. This signal is used to determine if the AIL has locked to a specific location in the delay chain.
- **Q[N:0]** - This geared down data which is now on the low speed Sclk. The serial bits are loaded into the parallel data starting with bit 0.
- **Valid** - This async signal is used to identify the degree of movement in the AIL delay chain.

LatticeSC Architecture Rules when Using AIL

The LatticeSC I/O logic architecture is captured inside a Programmable I/O Cell (PIC). Each PIC supports four Programmable I/Os (PIOs) labeled as A, B, C, and D. PIOs A and B combine to make a differential pair with A being the true and B being the complement. Likewise, PIOs C and D combine to make a differential pair with C being the true and B being the complement.

There is a single instance of the AIL per PIC. If using a differential input, the AIL can receive data on A/B or C/D. If using single ended data the AIL can receive data on A or C. If the A/B pair is using the AIL, then C/D can not use the AIL. C/D can be used as a general purpose I/O. However, if C/D are using the AIL, A/B can not be used as an input to the device and is only available for use as a general purpose output. This allows each PIC to be used as one input differential pair and one output differential pair.

The AIL is available in I/O Banks 2, 3, 4, 5, 6, and 7. The AIL is not available in I/O Bank 1. The AIL can be used with any of the LatticeSC input buffer standards including both single and differential inputs. For more information on the LatticeSC I/O buffer and I/O Banks please refer to technical note TN1088, *LatticeSC PURESPEED I/O Usage Guide.*

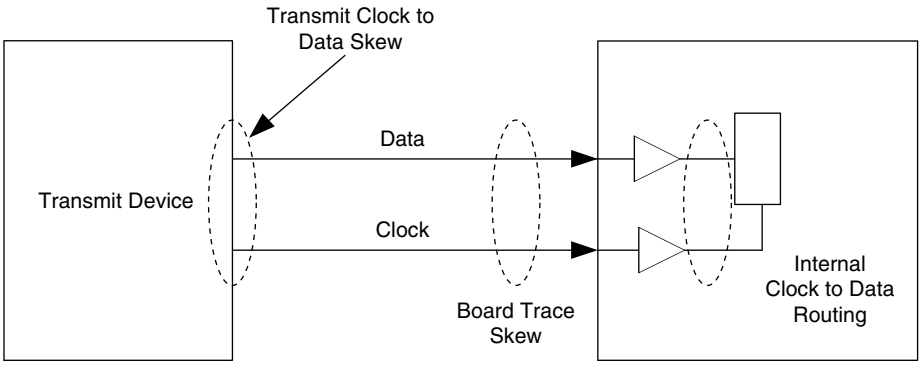
Applications

The LatticeSC AIL solution allows high speed input data to be sampled correctly. The following sections discuss different AIL applications.

Clock Forwarded Interfaces

I/O interfaces are commonly designed using a clock forwarded source synchronous architecture. With this architecture a transmitting device will provide a clock-to-data relationship on the forwarded clock. The board designer will need to leverage the transmitting devices output skew of data and clock to create a trace layout that will provide a tolerable skew at the receiving device. The receiving device needs to balance the input delay of the forwarded clock and data to create a proper setup and hold at the first stage register. Figure 18 shows a model of a typical clock forwarded interface.

Figure 18. Clock Forward Interface Timing Considerations



When taking into account the trace skew, along with the setup and hold constraints of the receiving register, the amount of data sampling margin shrinks. Allowing for worst-case slow, worst-case fast, temperature, and voltage deviation finding a solution at data rates > 1Gbps becomes difficult if not impossible.

The AIL can resolve this problem by eliminating the concern from both the board designer and the FPGA designer of balancing the delays and margin of the clock and data. The AIL is given an arbitrary clock and data relationship and will add delay as necessary to register clean data at that instance in time. This function is performed over process, temperature and voltage conditions without the need for a specific training pattern.

In some system architectures, the AIL does not even need the actual high speed forwarded clock. In many cases the low speed source of the high speed clock is present on the board. This low speed clock can be used and multiplied locally at the receiver with a LatticeSC PLL to the match the high speed clock rate. This new high speed clock can be used as the source to the AIL to capture the data. As long as the high speed rates match the AIL will find a solution by delaying the data.

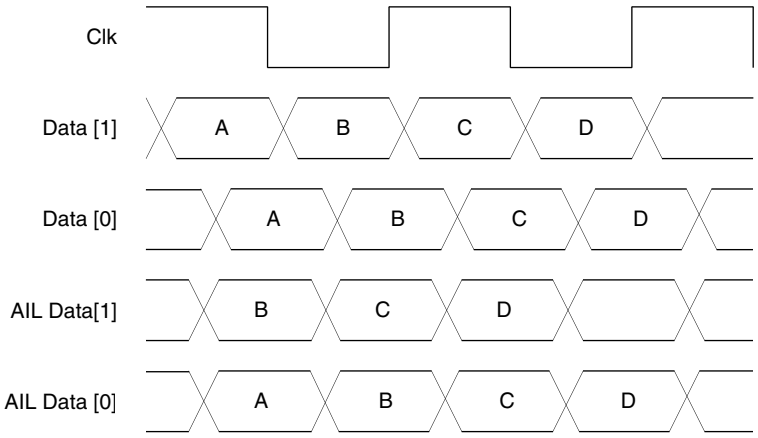
Processing the Serial Data

The AIL will sample the data and the I/O logic and provide a geared down version to the FPGA logic. Once in the FPGA fabric the design will need to take into account the serial nature of the data stream. Depending on the application of the FPGA this may include a word aligner and/or pattern detector. Implementation strategies for word alignment and pattern detection is beyond the scope of this document.

Bus Based AIL

The LatticeSC AIL can also perform the AIL function over multiple bits of an input data bus. When used in a bus format the AIL continues to solve the clock to data relationships on a bit per bit basis. Each bit is geared down in the I/O logic and the data presented to the FPGA fabric. As the AIL solves the clock to data relationship it is possible that the data bits may be off a clock cycle from each other across the bus. Figure 19 shows a waveform diagram of a 2-bit bus.

Figure 19. 2-Bit Bus AIL Example

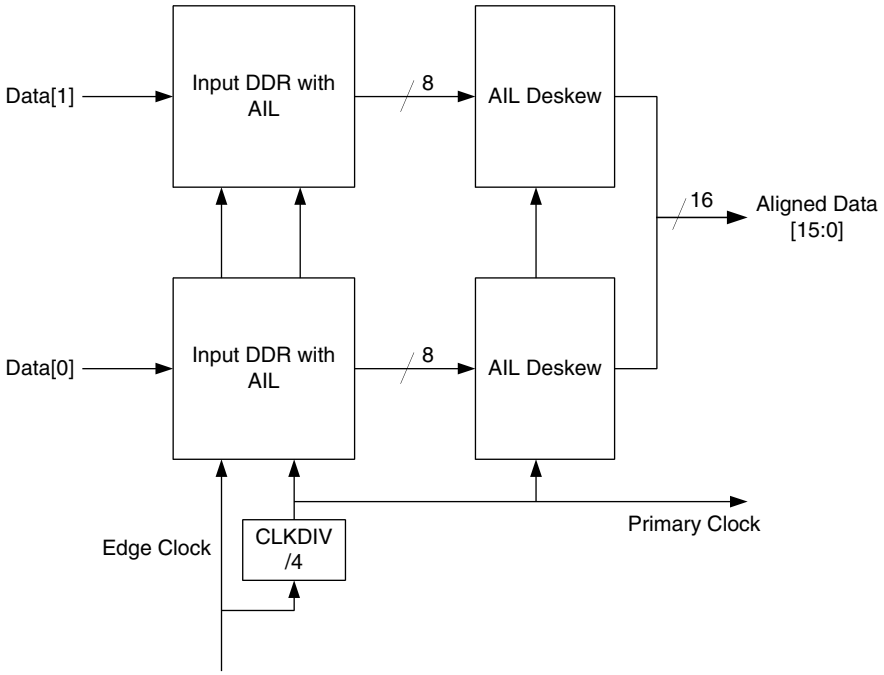


As shown in the waveform Data[1] and Data[0] contain the same data. Data[0] lags Data[1] by a small portion of the clock period. The AIL solves each bit separately and determines a clean location for the window. The clock-to-data relationship for each was solved differently for each bit of the bus. The final result is that the data is no longer aligned to the same clock edge that it was transmitted. To solve this problem Lattice has created the AIL Deskew block.

AIL Deskew

The AIL Deskew block is a FPGA based IP core which reassembles a multi-bit bus using a programmable training pattern. The AIL itself does not require a training pattern to sample correct data. In order to align all of the bits of the bus back to a common clock edge a training pattern is required to be sent on each bit. Once the training pattern has been identified, the AIL Deskew block will align all of the bits of the bus to a common clock. Figure 20 provides a block diagram of a 2-bit AIL bus using AIL Deskew.

Figure 20. LatticeSC AIL 2-bit Solution with AIL Deskew



A separate AIL Deskew block is instantiated for each input DDR module. By chaining several AIL Deskew blocks together a bus of arbitrary width can be created. The AIL Deskew block is available from Lattice for use in LatticeSC devices.

AIL Bus Training

The LatticeSC AIL does not require a specific training pattern for finding a location of the AIL window relying completely on arbitrary data transitions. When the RunAil input port of the AIL module is high the AIL will adjust the window accordingly. It may not be desirable for the AIL to continually update the window location. Specifically in the case of a bus, it is possible for one bit of the bus to be adjusted so that it becomes a clock cycle off from the other bits of the bus. This can be due to temperature, voltage, or other effects such as low frequency jitter. For this reason, when working with a bus it is best to de-assert RunAil when transmitting normal data. During the dedicated training pattern RunAil should be asserted to update the training for bus alignment.

The OIF Implementation Agreement for SPI4-02.1 provides an example of a dedicated data and dedicated training period. In this implementation bus training occurs after a number of user defined data bytes are transmitted (DATA_MAX_T). By constantly updating the training on the bus the specification allows the devices to accommodate for system changes that could alter the clock-to-data relationship across the bus. Lattice's SPI4.2 IP solution uses the AIL and dedicated logic to perform the training pattern portion of the design.

LatticeSC AIL Resources

Lattice provides the following resources for understanding more about the LatticeSC FPGA and I/O structure.

- TN1088, LatticeSC PURESPEED I/O Usage Guide
- LatticeSC/M Family Data Sheet
- IPUG44, LatticeSCM SPI4.2 MACO Core User's Guide

For information on using and demonstrating the LatticeSC AIL the following reference designs and demos are provided:

- LatticeSC FPGA 2Gbps Differential I/O Demo
- LatticeSCM SPI4.2 MACO Core

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
June 2007	01.0	Initial release.
November 2007	01.1	Updated Overview of the Adaptive Input Logic text section.
April 2008	01.2	Updated Acquisition Time text section.
		Removed Burst Mode Receiver section.
June 2010	01.3	Removed references to PURESPEED I/O Alignment reference design.