XAPP872 (v1.0) April 28, 2009

Creating a Controllable Oscillator Using the Virtex-5 FPGA IODELAY Primitive

Author: Martin Kellermann

Introduction.

This application note describes how to use the Virtex®-5 FPGA input/output delay (IODELAY) primitive as a means to create a high-precision adjustable oscillator with a wide tuning range.

Three different use models are described for the adjustable oscillator:

- Use as a delay-locked loop (DLL) for multiplication of a given reference clock.
- Use as a jitter cleaner for clocks with a high amount of cycle-to-cycle jitter.
- Use as a clock recovery circuit.

Background

Every I/O block of a Virtex-5 FPGA contains a programmable, absolute delay primitive called IODELAY. IODELAYs are a delay-chain of 64 taps with an average resolution of 80 ps per tap and are permanently calibrated using a reference clock of 200 MHz ± 10 MHz. The delay-chain is accessible from both the I/O side and the FPGA logic, and the number of used taps can be dynamically switched glitch-free at run time (see DS202, *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*, and UG190, *Virtex-5 FPGA User Guide)*.

The glitch-free switching capability and the accessibility from the FPGA logic allow for the creation of a basic oscillator circuit (Figure 1). The basic oscillator has a fixed number of taps and is very limited.

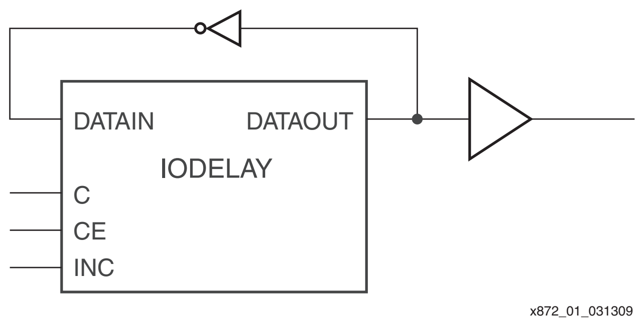


Figure 1: **Basic Fixed-Frequency Oscillator Using an IODELAY and an FPGA Inverter**

To extend the functionality of the basic fixed-frequency oscillator to allow frequency adjustment, the number of used taps needs to be permanently adjusted at runtime. When the oscillator is switched between two different tap values, it is switched between two different discrete frequencies. Control logic implemented in the FPGA logic guarantees that on average 0 ppm offset to the desired output frequency is maintained during switching. The Reference Design shows an implementation of this control logic.

Design Description

The IODELAY-based oscillator is extended to the model of a multiplying delay-locked loop (DLL), working in multiple stages. Based on the target frequency given at design time, the required number of taps is set at startup and reset. The loaded number of taps is then incremented or decremented to achieve the correct number of taps for the target frequency.

The target frequency needs to be a power-of-two multiple of the reference clock. The tested multiplication factors range from 1 to 32.

Two different control circuits are used to modify the number of taps used:

- A coarse loop using two binary counters, each running off either the reference clock or the DLL clock.
- A system based on FIFO control logic which maintains an exact match (average of 0 ppm frequency offset) between the multiplied DLL clock and the provided reference clock.

Coarse-Loop Adjustment

The coarse-loop adjustment of the DLL is run at startup and after reset. Two counters (their width dependant on the chosen DLL multiplication factor) are enabled. When the reference counter reaches its configured maximum value, it halts, sending a disable signal to stop the DLL counter. At this point, the counter values are compared in the clock domain of the DLL counter to ensure that the two counter values are stable for at least one clock period of the DLL clock. If the DLL counter value is higher than the reference clock counter multiplied by the DLL multiplication factor, the DLL frequency is too high, and tap value is decremented. If the DLL frequency is to slow, the tap value is incremented. After the tap adjustment is performed, both counters are reset and re-enabled in their clock domain.

The comparisons between the two counters are performed until the adjustment just overshoots, detected by setting flags to signal which adjustments have been performed. After both adjustments, speed-up and slow-down, are performed, the DLL is at one of the two correct operational taps, and the control circuit switches over to the DLL's fine loop using a FIFO control circuit.

Figure 2 shows a high-level diagram of the coarse-loop and fine-loop circuits.

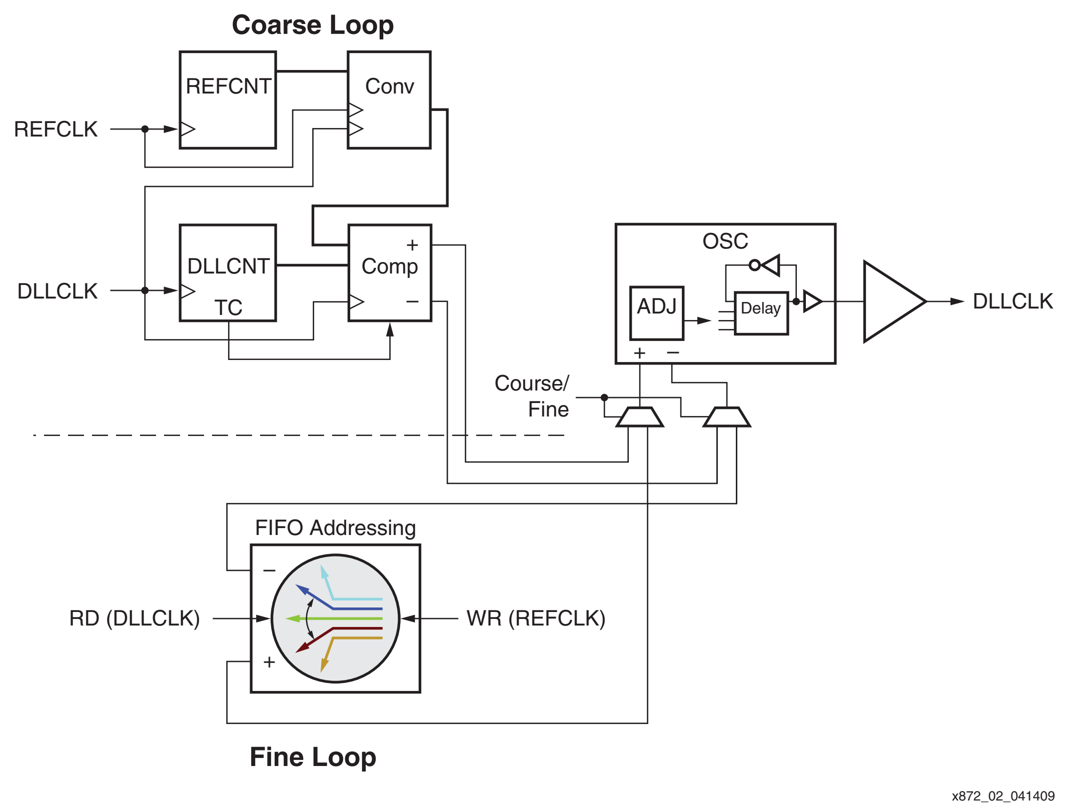


Figure 2: **Simplified Diagram of Coarse-Loop and Fine-Loop Circuit**

Fine-Loop Adjustment

The control circuit that compares the generated frequency versus the reference clock is based on the FIFO control logic implemented in the FPGA (Figure 3). This control logic is only the addressing logic of a FIFO, based on the structure of a write pointer and a read pointer, the FIFO memory itself is not needed and omitted. Read and Write FIFO operations only refer to operations on the address pointers. No data is stored or retrieved during these operations.

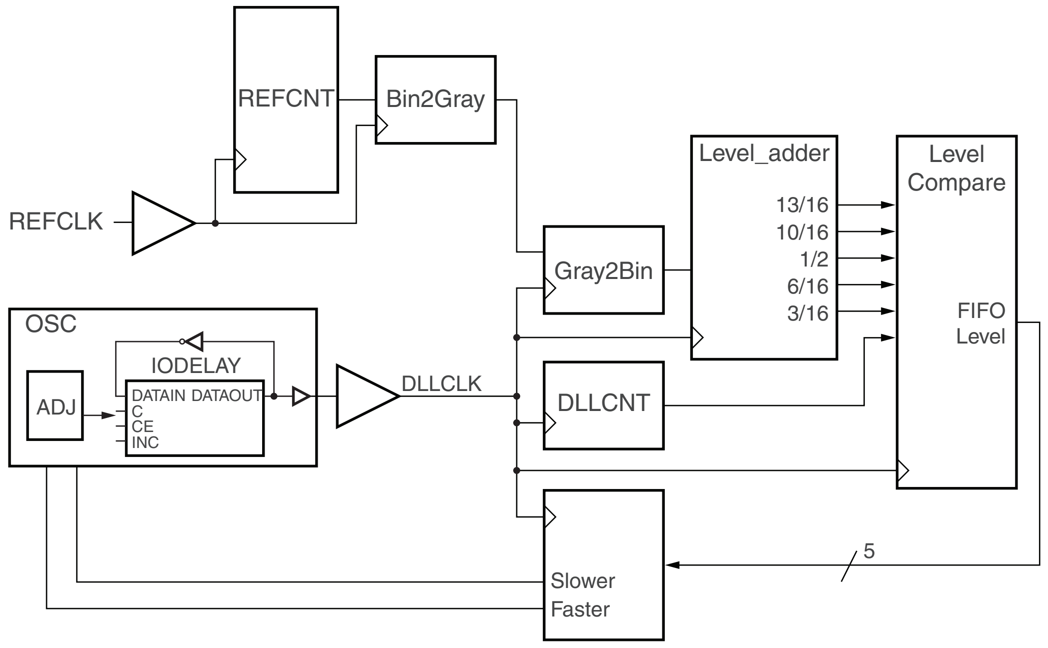


Figure 3: **FIFO Control Logic for an IODELAY Oscillator**

The output of the reference clock counter is Gray coded before crossing into the DLL clock domain to minimize the impact of errors. Inside the DLL clock domain, the value is converted back to binary. Then five offset numbers are added to the transferred value and compared to the value of the second counter, which is clocked by the output of the basic oscillator. These offset numbers allow the detection of five different FIFO fill levels.

Note: The Gray coding is done to avoid transferring a very wrong value into the DLL clock domain. For example, if the reference clock counter switched from `b0111` to `b1000`, due to circuit delays, the output could cross the value of `b1011` before reaching the final value. The Gray coding allows the transfer with a maximum error of 1 bit.

The five fill levels checked are 3/16, 6/16, 8/16, 10/16, and 13/16 of the address range. In the locked condition, the goal is to stay close to the half-full condition. As the FIFO empties, the flag for hitting 6/16 is asserted, and the tap number increments by one tap to slow down the generated clock. With this now slower clock, the FIFO fills up again and hits the mark 10/16, causing the delay line to decrement by one tap to speed up the clock (and causing the FIFO to empty). When the address hits the 6/16 mark again, the tap number increments again and so on. This leads to switching between two discrete frequencies which are slightly faster and slightly slower than the target frequency, but are, on average, the right frequency. If this generated clock is examined using an oscilloscope, it appear to be *running away* (Figure 11).

The FIFO marks of 3/16 and 13/16 are used in case the frequency drifts and for ensuring that the exact frequency is hit at the beginning of the fine-loop adjustment. With a stable reference clock, these two marks are typically not used after the DLL is locked, but can be used in the case of frequency drift of the reference clock.

Figure 4 shows how the number of taps is adjusted after start-up. In this case, the target frequency is 149 MHz, and 42 taps are loaded into the IODELAY at start-up. Over several adjustments, the actual frequency is increased and overshoots at T = 7. At this point, the control circuitry switches to the fine-loop adjustment.

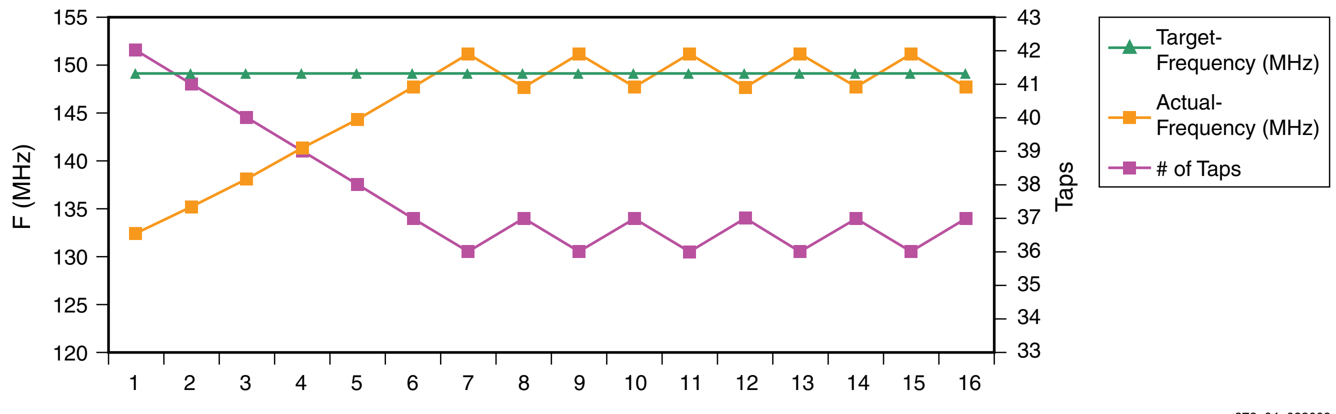


Figure 4: **Coarse and Fine-Loop Adjustments**

DLL Lock Time

The lock time of the DLL depends on how long it takes to reach the correct target tap from the configured initial tap entered at design time. The lock time is based on:

- Configured start tap
- Target value of tap
- Number of tap adjustments
- Period of the reference clock

Table 1 provides suggested start taps and divider factors over the supported frequency range. When using, these start taps and divider factors are calculated based on the two generics, TARGET_FREQUENCY and DLL_MULTIPLY, available in the top-level design in the Reference Design file. The calculations are performed using the package file `pkg_dll_reco_r1.0.(vhd/.v)`.

Table 1: **Start Taps and Division Factors for Supported Output Clock Frequencies**

	Output Frequency Ranges (in MHz)								
	12–24	**25–54**	**55–74**	**75–109**	**110–149**	**150–174**	**175–199**	**200–224**	**225–250**
Divider Factor	8	4	2	2	1	1	1	1	1
Start Taps	40	33	42	28	42	32	27	23	20

Lock time can be calculated as follows:

Locktime = AdjustmentSteps × {Ceiling (100/DLLMultiply,1) + DomainCrossing} × {TargetPeriod × DLLMultiply}

With:

TargetPeriodOsc [ns] = (1/TargetFrequency [MHz] × 1000) / DividerFactor

TargetTaps = (TargetPeriodOsc[ns] – 2 ns) / (2 × 0.078 ns)

AdjustmentSteps = Ceiling(Abs(TargetTaps-StartTaps)) + 1

DomainCrossing is an empirical value and depends on DividerFactor:

Divider factor = 1 → DomainCrossing = 10

Divider factor = 2 → DomainCrossing = 8

Divider factor ≥ 4 → DomainCrossing = 4

As the lock times vary, depending on desired target frequency and DLL multiplication factor, these calculations are made available in the Reference Design file as an Microsoft Excel spreadsheet, `locktimes.xls`. The first worksheet, LockTime, is used to calculate the lock time, based on values set in the second worksheet, StartTap. The usage of LockTime and StartTap is explained in the spreadsheet.

Design Capability and Range

The reference clock (REFCLK) can be provided either externally or internally and is multiplied by a factor of between 1 and 32 (powers of 2 only) defined at design time.

The frequency of DLLCLK depends on the number of taps used by the DLL. The minimum frequency of the tap delay line is with all 64 taps used plus the delay of the inverter loop which is approximately 1 ns. With an average delay per tap of 80 ps, the maximum period of this clock is:

$$T_{MAX} = 2 \times (64 \times 80\text{ps} + 1\text{ns}) = 12.24\text{ns} \qquad \textit{Equation 1}$$

where

$$f_{MIN} = \frac{1}{T_{MAX}} = 82\text{MHz} \qquad \textit{Equation 2}$$

The minimum clock period is artificially limited to seven taps to avoid very high frequencies:

$$T_{MIN} = 2 \times (7 \times 80\text{ps} + 1\text{ns}) = 3.12\text{ns} \qquad \textit{Equation 3}$$

where

$$f_{MAX} = \frac{1}{T_{MIN}} = 320\text{MHz} \qquad \textit{Equation 4}$$

The limit of seven taps was chosen based on the maximum clock frequency of the Virtex-5 FPGA BUFR networks of between 250 MHz and 300 MHz, depending on the speed grade. The value of seven taps leaves some margin above the BUFR specification. While the intent is to avoid creating a narrow design specification or a new data sheet specification, a prudent designer must assure that the maxiumn BUFR frequency is not violated.

If output clock frequencies lower than F_{MIN} are necessary, then either dividers built out of binary counters directly behind the oscillator or the division feature of the BUFR can be used. For the reference design, only the binary counters are used for simplicity.

Use Models

Clock Multiplier

An obvious application of the IODELAY oscillator is as clock multiplier is, providing an output that is power-of-two multiple of the reference frequency.

Jitter Cleaner

Another application of the IODELAY oscillator is as a jitter cleaner or definer. In this application, the multiplication factor is set to 1, generating an independent clock that on average has the correct frequency. The jitter of this clock is only dependent on the switch between taps and completely independent of the jitter on the reference clock. The maximum expected cycle-to-cycle jitter of the generated clock is mainly dependent on the switching between taps. The worst-case jitter due purely to switching in the native operation range of 110 to 250 MHz is two times the delay of one tap, or 160 ps onaverage.

Additional jitter sources can also affect the generated clock, so cycle-to-cycle jitter of approximately 200 ps can be expected. When an operational range is chosen using the dividers, the output jitter of the divided clock does increase in steps. This effect is caused by the fact that multiple clock periods are *accumulated* by the dividing counter before the most significant bit, the divided clock, switches. Figure 5 shows a situation where the output clock is generated by a divide by 4 and where the original clock switches between 21 and 22 taps (routing delay inside the FPGA is ignored).

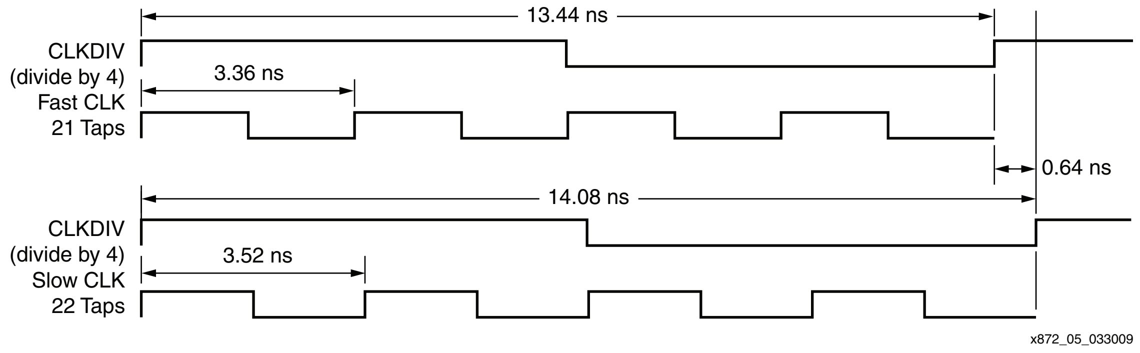


Figure 5: **Accumulated Phase Difference Resulting from Clock Dividers**

The worst-case cycle-to-cycle jitter occurs when the IODELAY oscillator switches from one discrete frequency to another (from one tap value to the next) at the same time as the dividing counter restarting to zero. This situation causes the MSB of the counter to be toggled with two different periods (Figure 5).

With this situation as example, the period for the fast clock (at tap 21) is:

21 Taps $\times$ 80 ps $\times$ 2 = 3.36 ns

With a divider factor set to 4, the output clock period is:

T_{OUTPUT_CLOCK} = 3.36 ns $\times$ 4 = 13.44 ns

For the slow clock (at tap 22), the period is:

22 Taps $\times$ 80 ps $\times$ 2 = 3.52 ns

Again, with a divider factor set to 4, the output clock period is:

T_{OUTPUT_CLOCK}= 3.52 ns $\times$4 = 14.08 ns

The resulting maximum cycle-to-cycle jitter is equal to:

$JITTER_{MAX}$ = 14.08 ns – 13.44 ns = 0.64 ns (or 4 $\times$ 160 ps)

The calculations show that when using a divider factor to reach lower operating frequencies, the expected worst-case cycle-to-cycle jitter is the normal switching difference of 160 ps multiplied by the divider factor used (Table 2).

Table 2: **Expected Cycle-to-Cycle Jitter with Different Divider Factors**

	Divider Factor			
	1	**2**	**4**	**8**
Expected Jitter (in ps)	160	320	640	1280

Using this approach, it is possible to improve the quality primarily of fast clocks if the two discrete frequencies can be tolerated. For lower frequencies, the cleaning effect is less; however, a modification of the design is possible. This modification involves creating a longer delay chain by cascading a second IODELAY set to the maximum delay with an adjusted IODELAY. With an estimated routing delay of 1 ns between the IODELAYs, and 1 ns for the inverter, the total clock period is extended to:

2 $\times$ (5 ns + 1 ns + 5 ns + 1 ns) = 24 ns or 41 MHz.

A block diagram of this modification is shown in Figure 6.

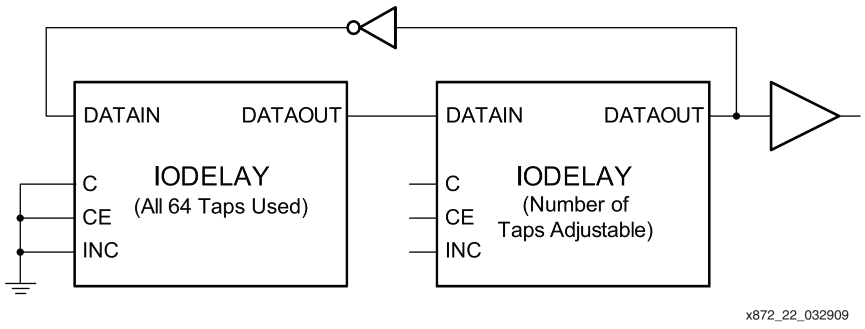


Figure 6: **Cascaded IODELAYs for Smaller Jumps at Lower Frequencies**

Clock Recovery

A third application is to use the IODELAY oscillator for clock recovery, for example for designs using oversampling to recover the data. These designs often deliver a clock together with an enable signal. The disadvantage of an enable-signal approach is that the delivered clock can be substantially higher than necessary, leading to tighter timing constraints for the downstream logic.

One way around a gapped or enabled clock is to use the delivered clock and its enable signal as reference clock and on the write enable of the FIFO logic. The write enable in the other application is asserted after reset of the IODELAY oscillator and remains active. For clock recovery with enabled clocks, the enable signal of the FIFO control can be ANDed together with the enable from the recovery block to provide the correct amount of clock cycles as input. The output of the IODELAY oscillator is the two known frequencies averaging to the correct overall frequency.

Testing the IODELAY Oscillator

The IODELAY oscillator design described in this application note has been tested in several devices and on several demonstration boards:

- The Virtex-5 LXT FPGA ML505 Evaluation Platform with a Virtex-5LX50TC-1
- The Virtex-5 FPGA RocketIO™ Characterization Platform Characterization Platform ML523 with a Virtex-5 LX110TC-1
- AFX Board with Virtex-5 SX95TC-1

The Test Design

To test the capabilities of the IODELAY oscillator, the design shown in Figure 7 is used. The externally provided reference clock is routed to a DCM, where a clock multiplication of 33/32 takes place for a high amount of jitter on the CLKFX output and then to a BUFGMUX where the unaltered input clock is present in parallel. This configuration allows for the choice between a clock with a high amount of jitter and a standard clock. The output of the first BUFGMUX is routed to the input of the IODELAY oscillator, a second DCM, and directly to a BUFGMUX. The BUFGMUX selects between the CLKFX output of the second DCM, and the output of the first BUFGMUX.

Note: This routing is necessary to avoid the situation of having a CLKFX output driving a second DCM.

Depending on the target frequency, the DCMs are generated in low or high-frequency mode. All implementations of the test design are constrained for the maximum frequency of 250 MHz. Although this frequency does not apply to all designs, it results in a single UCF file for all implementations. Timing violations do occur because of this setting; however, the resulting timing is still within the necessary range to test the designs.

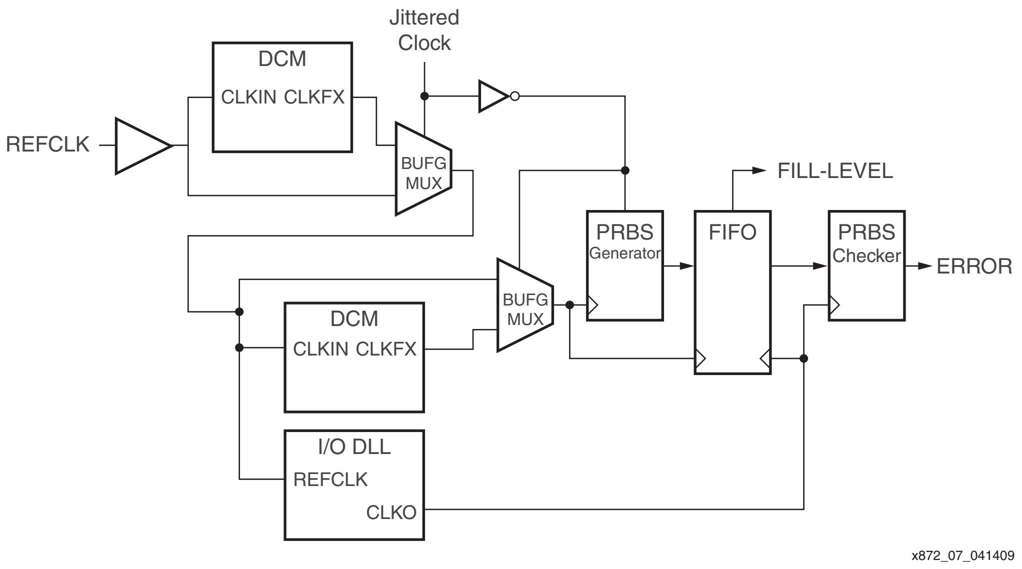


Figure 7: **Test Design for Jitter Cleaner and Clock Multiplication**

When the clock multiplication is to be tested, the first DCM is bypassed and the second DCM has the same multiplication factor as the IODELAY oscillator. A PRBS generator is driven by the multiplied CLKFX output and writes into a standard FIFO. The IODELAY oscillator multiplies to the same nominal frequency as the DCM and reads from the FIFO. The data from the FIFO is then checked by a PRBS checker with the same polynomial as the generator.

Once both the DCM and the IODELAY oscillator are locked and hence their average frequencies should be the same, errors can only occur if the IODELAY oscillator does not have an average offset of 0 ppm to the target frequency, causing an under or overflow of the FIFO. The error signal of the PRBS checker is used as a trigger signal for the ChipScope™ analyzer to show if a non-0 ppm situation occurs.

In addition to the offset, the fill level of the FIFO between the two clock domains is monitored. The expect output should appear similar to Figure 4 when both the DCM and the IODELAY oscillator are locked.

For these tests, the multiply and divide values shown in Table 3 are used. The multiply values are specified by the user, where divide values are based on the targeted output frequency.

To test the M = 1 the jitter cleaner example, the CLKFX output of a Virtex-5 FPGA DCM is used with the DCM multiplication set to 33, and the DCM division set to 32. This results in a clock with a high amount of jitter, demonstrating the jitter cleaning capability of the IODELAY oscillator. For all other M and D combinations, an external clock generator as described in Frequency Generator for Spartan-3E Starter Kit is used as a clock source. For the tests both the lower and the upper frequencies are used.

Table 3: **Input Frequency Ranges for Test of M and D Values**

		M							
		1		2		8		32	
		FIN$_{MIN}$ (MHz)	FIN$_{MAX}$ (MHz)	FIN$_{MIN}$ (MHz)	FIN$_{MAX}$ (MHz)	FIN$_{MIN}$ (MHz)	FIN$_{MAX}$ (MHz)	FIN$_{MIN}$ (MHz)	FIN$_{MAX}$ (MHz)
D	1 (FOUT$_{MAX}$ = 250 MHz)	110	250	55.5	125	13.75	31.25	3.44	7.81
	2 (FOUT$_{MAX}$ = 109 MHz)	55	109	27.5	54.5	6.88	13.63	1.72	3.41
	4 (FOUT$_{MAX}$ = 54 MHz)	25	54	12.5	27	3.13	6.75	0.78(1)	1.69
	8 (FOUT$_{MAX}$ = 24 MHz)	12	24	6	12	1.5	3	0.382(1)	0.752(1)

Notes:

1. These corner values are only tested for the output jitter but not for 0 ppm offset due to the higher minimum input frequency of the DCM.

Test Design Implementation

The measurement numbers are based on multiple implementations of the design shown in Figure 7. The test design is provided in both VHDL and Verilog, where the Verilog version is a translation of the VHDL. In addition, batch files for synthesis and implementation of the test suite are included in the design archive. Each board directory contains a batch file called `synth.bat` used for running multiple synthesis runs with different M and D values. Each tested M and D value has its own sub-directory where the implementation data is stored. Implementation is called by starting the batch file `implement.bat`. The resulting BIT files are copied over to the sub-directory `/bits` and carry the name of their M and D values (for example, `M1D1.bit`).

The synthesis run uses the design data from the VHDL directory. The Verilog design is provided for convenience.

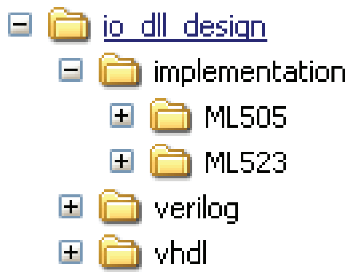


Figure 8: **Simplified Directory Structure of Test Design**

Test Method

Following measurements are performed:

- Check the 0 ppm average offset to target frequency via the FIFO
- Measure of cycle-to-cycle jitter
- Take screenshots of the two discrete frequencies
- Measure and compare the FFT of reference clock frequency and output frequencies

Jitter Cleaner Test Results

For this test the multiplication factor of the IODELAY oscillator is set to 1, and the input and output cycle-to-cycle jitter is measured at both the low and the high limit of the operating ranges. Test results are given in Table 4.

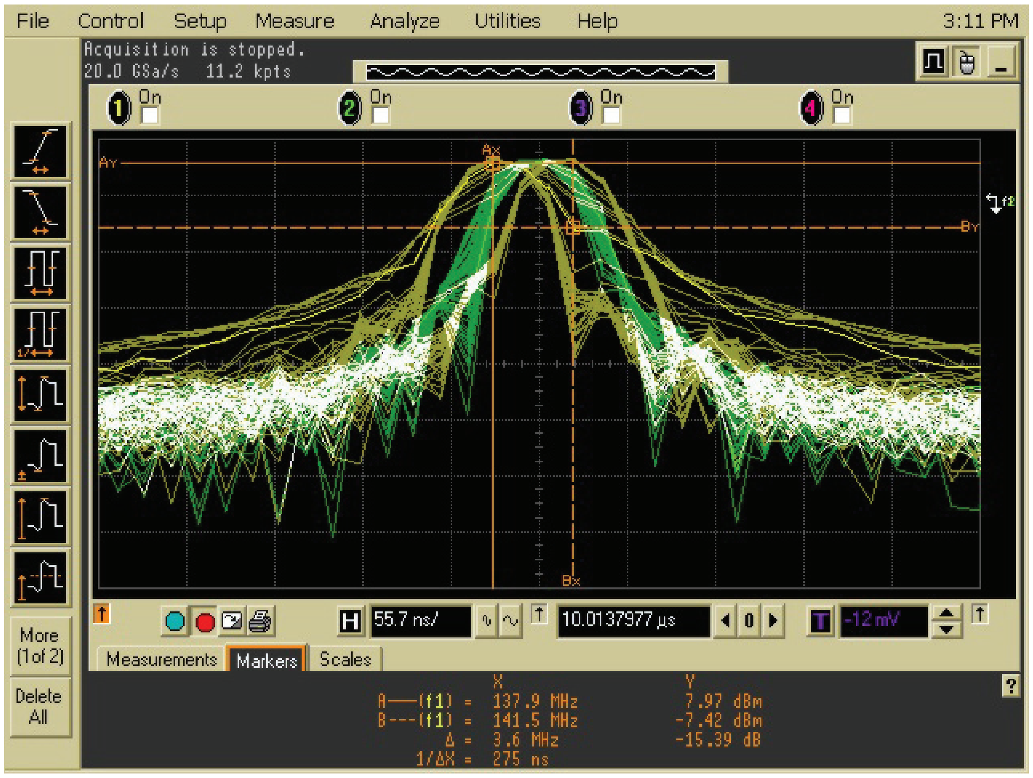
Table 4: **Cycle-to-Cycle Jitter for Input (CC_I) and Output (CC_O) Clocks in Jitter Cleaner Mode**

	D								Units
M = 1	**8**		**4**		**2**		**1**		
Measured on ML505 Board									
Frequency	14.5[1]	24	25	54	55	109	110	250	MHz
CC_I	8070	4600	5370	821	1400	326	290	159	ps
CC_O	1000	1160	679	502	323	216	180	131	ps
Measured on ML523 Board									
Frequency	16[2]	24	25	54	55	109	110	250	MHz
CC_I	8400	5270	3900	944	1380	332	334	206	ps
CC_O	1120	1150	634	555	370	249	178	178	ps

Notes:

1. 14.5 MHz was used as minimum frequency as the jitter generation DCM in the present configuration did not lock at lower frequencies.
2. 16 MHz was used as minimum frequency as the jitter generation DCM in the present configuration did not lock at lower frequencies.

Figure 9 shows the original 140 MHz clock frequency in green and the two discrete frequencies with a slightly higher and slightly lower frequency in yellow. The original frequency is located somewhere between the two frequencies generated by the IODELAY oscillator.



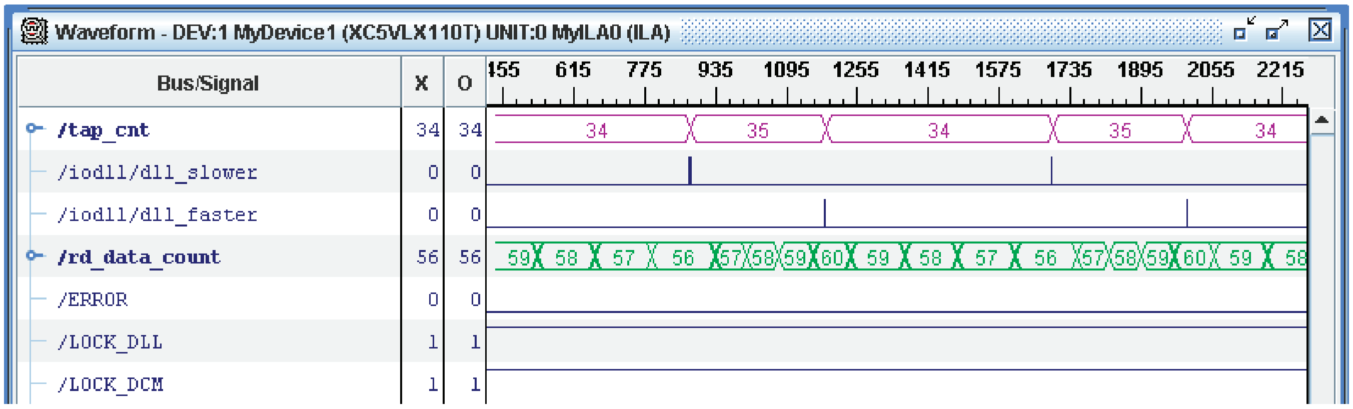
```
Acquisition is stopped.
20.0 GSa/s   11.2 kpts
```

```
                X                Y
A——(f1) =   137.9 MHz        7.97 dBm
B---(f1) =  141.5 MHz       -7.42 dBm
       Δ =  3.6 MHz        -15.39 dB
    1/ΔX =  275 ns
```

x872_09_033009

Figure 9: **FFT of Input and Output Clock in Jitter Cleaner Mode at 140 MHz**

Figure 10 shows the switching between the two taps to create the output clocks. The IODELAY oscillator stays at 34 taps longer than at 35 taps, which is consistent with Figure 9 with the input frequency being slightly closer to the upper frequency than to the lower.

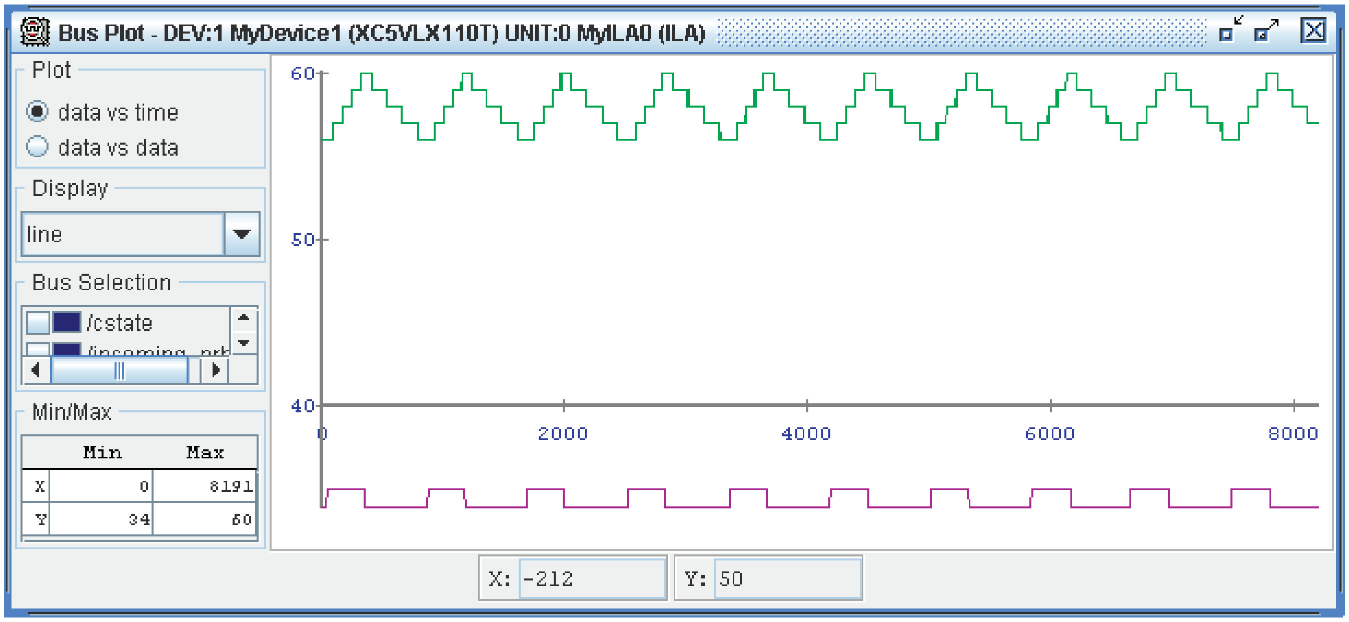

Waveform - DEV:1 MyDevice1 (XC5VLX110T) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
/tap_cnt	34	34
/iodl1/dll_slower	0	0
/iodl1/dll_faster	0	0
/rd_data_count	56	56
/ERROR	0	0
/LOCK_DLL	1	1
/LOCK_DCM	1	1

x872_10_033009

Figure 10: **ChipScope Analyzer Waveform of Shift Between Two Taps for Operation at 140 MHz**

Figure 11 shows the distance between the write and the read pointer of the FIFO between the original clock and the one generated by the IODELAY oscillator. As expected a repeating triangular waveform (green) appears, moving around a fixed number of entries. At the bottom of the diagram, the number of taps (purple) is shown, again consistent between the number of taps and the direction of the distance difference.


Bus Plot - DEV:1 MyDevice1 (XC5VLX110T) UNIT:0 MyILA0 (ILA)

Plot: [✓ data vs time] data vs data

Display: line

Bus Selection: /cstate, /incoming_prb

Min/Max

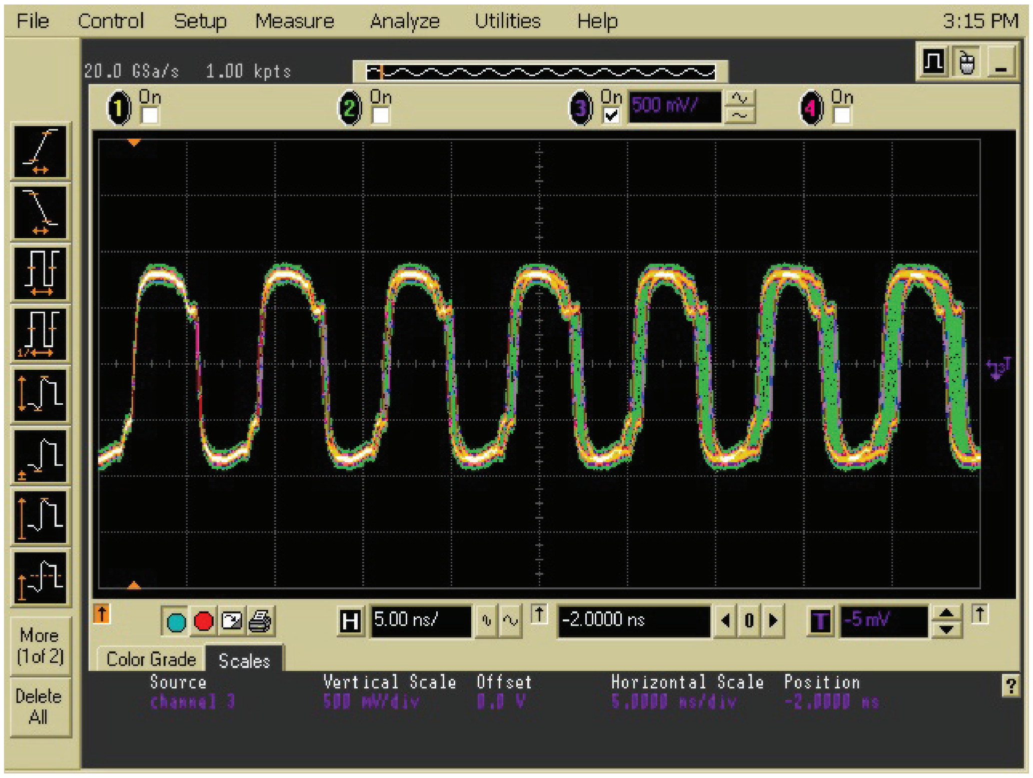
	Min	Max
X	0	8191
Y	34	60

X: -212 Y: 50

x872_11_033009

Figure 11: **ChipScope Analyzer Bus View of FIFO Distance Between DCM Clock and DLL Clock and Tap Numbers**

Figure 12 shows the generated IODELAY oscillator in the time domain. The trigger is set on the generated clock, with multiple clocks recorded with the oscilloscope's persistence set to infinite. Over multiple clock cycles the effect of the two discrete frequencies becomes more apparent.



Source: channel 3
Vertical Scale: 500 mV/div
Offset: 0.0 V
Horizontal Scale: 5.0000 ns/div
Position: -2.0000 ns

Figure 12: **140 MHz Output Clock (Different Period Lengths are Visible on Right)**

Clock Multiplier Test Results

For this test the multiplication factors are set to the appropriate M values and the target frequency is set to halfway between the minimum and maximum values of the appropriate range. The input frequencies for the various multiply and divide options are shown in Table 3. The results correspond well with the effect shown in Figure 5 — higher maximum jitter at higher divider ratios.

Table 5: **Output Cycle-to-Cycle Jitter for Clock Multiplier Mode, Measured on an ML505 Board**

		D								**Units**
		8		**4**		**2**		**1**		
		F_{OUT} (MHz)		**F_{OUT} (MHz)**		**F_{OUT} (MHz)**		**F_{OUT} (MHz)**		
		10	**24**	**25**	**54**	**55**	**109**	**110**	**250**	
Measured on ML505 Board										
CC_O	M=2	865	877	517	591	308	211	166	170	
	M=8	841	923	547	635	374	396	258	171	
	M=32	848	936	492	621	326	315	274	143	

Table 5: **Output Cycle-to-Cycle Jitter for Clock Multiplier Mode, Measured on an ML505 Board *(Cont'd)***

		D								Units
		8		4		2		1		
		F_{OUT} (MHz)		F_{OUT} (MHz)		F_{OUT} (MHz)		F_{OUT} (MHz)		
		10	24	25	54	55	109	110	250	
Measured on ML523 Board										
CC_O	M=2	1400	1120	570	531	345	260	170	170	
	M=8	1200	1110	613	523	310	242	178	177	
	M=32	1370	1120	542	524	306	237	172	169	

Potential Issues and Counter Measures

High-Frequency Oscillation

A potential danger of the ring oscillator used in this application note is when a very short delay is rolled back to a very long delay. At first, this situation does not appear to be problematic as the delay line is just extended, and the oscillation is immediately damped (Figure 13). Initially a fast oscillation caused by the short delay of zero taps (Delay 1) is switched so that all taps are used, causing the long delay (Delay 2). An expected output of this oscillator is a fast oscillation followed by a slow oscillation (MUXOUT).

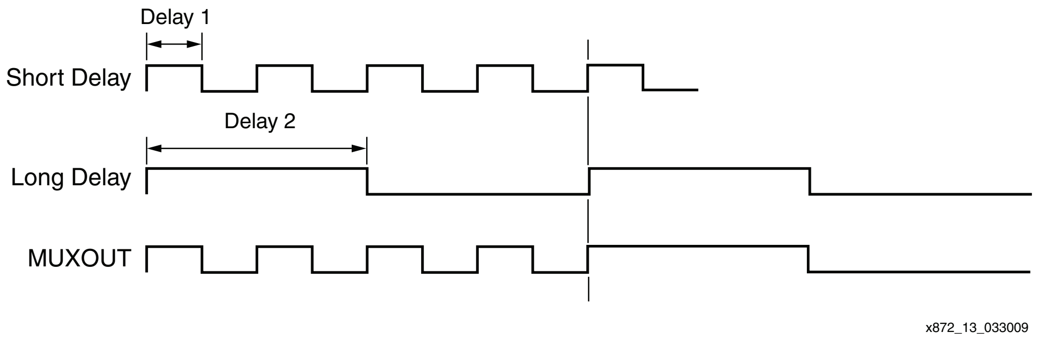


Figure 13: **Expected Oscillation after Roll Back from Short to Long Delay Line**

This assumed behavior is, however, incorrect. The reason is the way the IODELAY is built by using a series of delay taps where the outputs of the tap delays are fed into a multiplexer. Figure 14 shows a very simplified diagram of this concept.

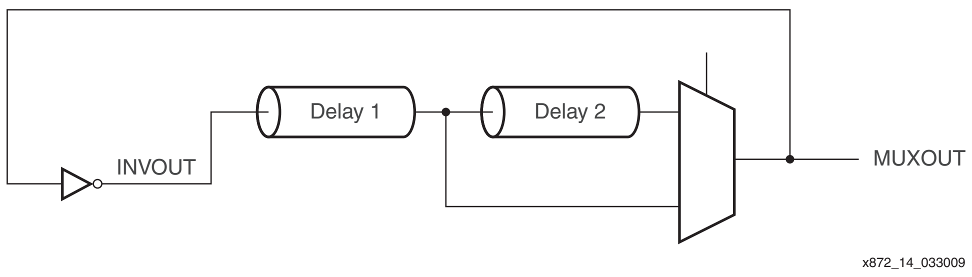


Figure 14: **Simplified Diagram of IODELAY Oscillator**

The short delay causes a very fast oscillation which is also present in the unused delay line. When the MUX is switched over to the long delay line during the fast oscillation, the fast oscillations are present as well. The only visible impact on the output MUXOUT is a potential phase shift, but the oscillation itself continues at full speed (Figure 15).

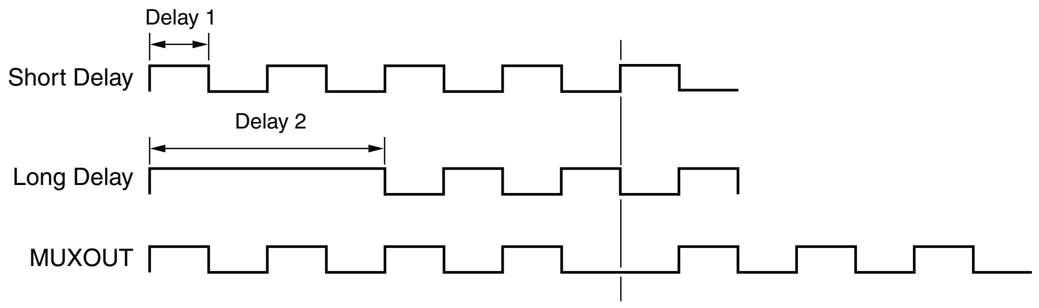


Figure 15: **Actual Oscillation after Roll Back of IODELAY Taps**

This potential issue is avoided in the IODELAY oscillator design by not allowing a rollover of the IODELAY taps in either direction. A minimum setting of seven taps is always maintained, and never more than the 63 taps are used.

Large Phase Jumps with High Divider Factors

When using the IODELAY oscillator for creating slower clocks by using the divider factors after the oscillator, relatively large phase jumps can occur (Table 2). If an application cannot tolerate these phase jumps, some modification can be made to the design (Figure 6). The oscillator can be modified to use a longer delay line by connecting two or more IODELAYs in series, and having one inverter in the chain.

The first IODELAY is sourced by the inverter on its DATAIN input with the Generic IDELAY_TYPE set to FIXED. The DATAOUT output of this IODELAY sources the next one, again on the DATAIN input. The last IODELAY connects to the input of the inverter, which feeds the initial IODELAY. The second and any successive IODELAYs have their IDELAY_TYPE set to FIXED. For all IODELAYs, the DELAY_SRC must be set to DATAIN.

Conclusion

The IODELAY oscillator is a reliable way of generating and cleaning clocks, avoiding the use of DCMs or PLLs. This capability can be useful for designs requiring a large number of clocks, and when the dedicated clock resources are in short supply or do not provide the appropriate functionality.

Reference Design

The design checklist in Table 6 includes simulation, implementation, and hardware verification details for the reference design:

https://secure.xilinx.com/webreg/clickthrough.do?cid=119746

Table 6: **Design Checklist**

Parameter	Description
General	
Developer Name	Martin Kellermann
Target Devices	Virtex-5 LXT/SXT/FXT FPGAs, production, all speed grades
Soruce Code Provided	Yes
Source Code Format	VHDL, Verilog
Design Uses Code/IP from Existing Application	Yes, Core Generator software with: • FIFO Generator Version 3.3 • Chipscope Pro Version 10.1
Simulation	

Table 6: **Design Checklist *(Cont'd)***

Parameter	Description
Functional Simulation Performed	Yes
Timing Simulation Performed	No
Testbench Used for Functional Simulation	Yes
Testbench Format	VHDL, Verilog
Simulator Software Version Used	Modelsim Version 6.0
SPICE/IBIS Simulation	No
Implementation	
Synthesis Software Tools/Version Used	XST, version 10.1 SP3
Implementation Software Tools/Versions Used	ISE software 10.1 SP3
Static Timing Analysis Performed	Yes
Hardware Verification	
Hardware Verified	Yes
Hardware Platform Used for Verification	Tested on following boards: • ML505 • AFX-board with Virtex-5 SX95T FPGA • ML523

Table 7 lists the device utilization and performance data.

Table 7: **Devices Utilization and Performance**

Parameter	Specification	Description
Maximum Output Frequency	-1 speed grade	F_{OUT}: • 320 MHz (when configured to use BUFG) • 250 MHz (when configured to use BUFR)
	-2 speed grade	F_{OUT}: • 320 MHz (when configured to use BUFG) • 250 MHz (when configured to use BUFR)
	-3 speed grade	F_{OUT}: • 320 MHz (when configured to use BUFG) • 250 MHz (when configured to use BUFR)
Device Utilisation without Test-Circuits	Slices	approx. 100 (depending on design configuration)
	BUFG	0/1 (depending on design configuration)
	BUFR	0/1 (depending on design configuration)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/28/09	1.0	DRAFT of Initial Xilinx release.

Notice of Disclaimer