Clock Reconstruction with Low-Cost External DCXO

2013.11.30

AN-695 Subscribe Send Feedback

You can use Altera FPGAs in applications where multiple client signals need a high-quality (low-jitter) transmit reference clock.

This application note describes a clock reconstruction solution that you can use to achieve this jitter compliance at a low total cost. The solution maintains a high degree of flexibility using an Altera FPGA and a cost-effective external Microsemi ZL30240 digitally controlled oscillator (DCXO).

This solution can meet the jitter requirements for conformance with the following relevant protocols:

- SDH/SONET
- Serial digital interface (SDI)
- Optical transport network (OTN)

This application note assumes you have experience designing clock reconstruction circuits for transmission networks or similar applications.

For more information on PLL theory that this application note uses, refer to *Phaselock Techniques*, Floyd M. Gardner (3rd edition 2005).

Features

- Simple serial peripheral interface (SPI) to external DCXO component
- CPU access for DCXO configuration
- Live DCXO updates to track reference clock
- Flexible, fully programmable, proportional plus integral (PI) controller
- Infinite impulse response (IIR) filter for jitter reduction

Background

Clock reconstruction is important in many applications, where you need to generate a physical reconstructed clock based on incoming timing information.

The incoming timing information can be:

- An actual incoming clock
- A gapped clock signal
- Arriving data
- A timing protocol (e.g. IEEE 1588)
- A proprietary timing signal (e.g. bitcounts, timestamps, etc.)

The reconstructed clock must typically conform to several specifications, for example:

- Jitter specifications
- Wander specifications

The control circuitry that generates the required clock might have some characteristics that must fall within required limits:

- Tracks incoming timing within a pull range (e.g. specified as maximum ppm offset)
- Attenuates jitter outside a given 3dB bandwidth
- Complies with jitter transfer peaking limit and damping

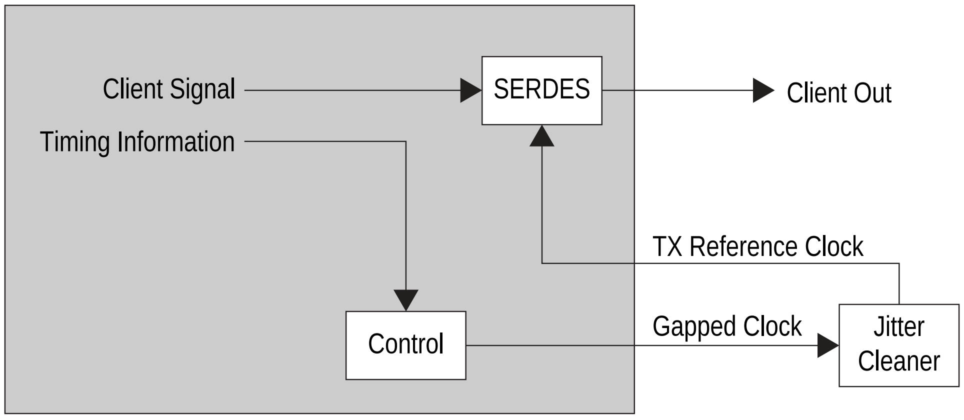
Potential Solutions

Two general classes of solution are the gapped clock and the DCXO-based solution.

Gapped Clock Solution

In the gapped clock solution, the FPGA uses the incoming timing information to generate a signal with an equivalent rate to the desired clock. The FPGA generates this clock in an unrelated clock domain, and it is a logical signal with the same number of transitions in a given period as the desired clock, but with varying mark and space ratio.

Figure 1: Gapped Clock



The FPGA generates the gapped clock as an open-loop signal (i.e. without using the reconstructed clock).

The FPGA then passes the gapped clock through an external jitter cleaner device, typically an integrated PLL and DCXO. The jitter cleaner device regenerates a true clock signal that tracks the gapped clock and thus the incoming timing information.

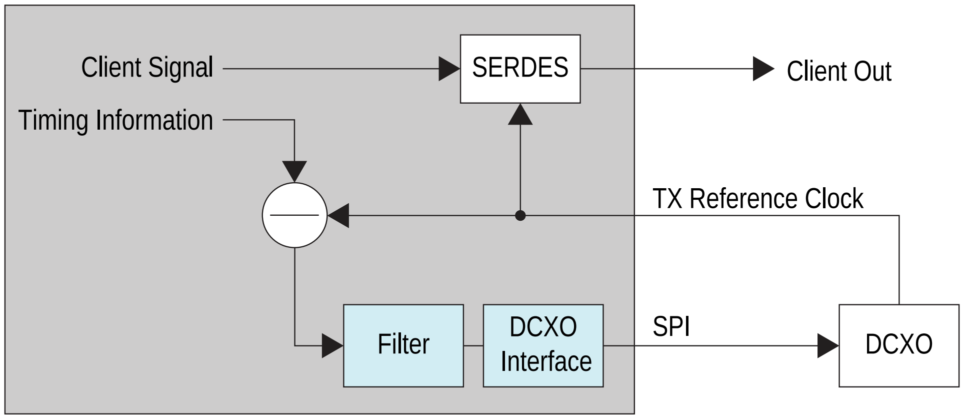
However, this solution has the following disadvantages:

- High cost, which is a significant factor in solutions with many independent clocks.
- Lack of flexibility. You can only configure the jitter cleaner within a limited performance range.
- Inability to modify filter behavior under transient or recovery situations.

DCXO-based Solution

In the DCXO-based solution, a simple external DCXO generates a clock based on a digital control signal on a SPI bus from the FPGA. A digital control loop, using the same principles as a PLL, varies the control input to the DCXO so that the generated clock tracks the timing information.

Figure 2: DCXO Control Loop



This solution can be very flexible, and results in a lower final cost, as the per-port cost of a jitter-compliant DCXO solution is lower than that of an equivalent jitter cleaner device.

The solution uses two IP cores: the DCXO interface and the Filter. These IP cores enable a standardized solution that you can use to recreate almost any reference clock based on timing information.

The DCXO Interface

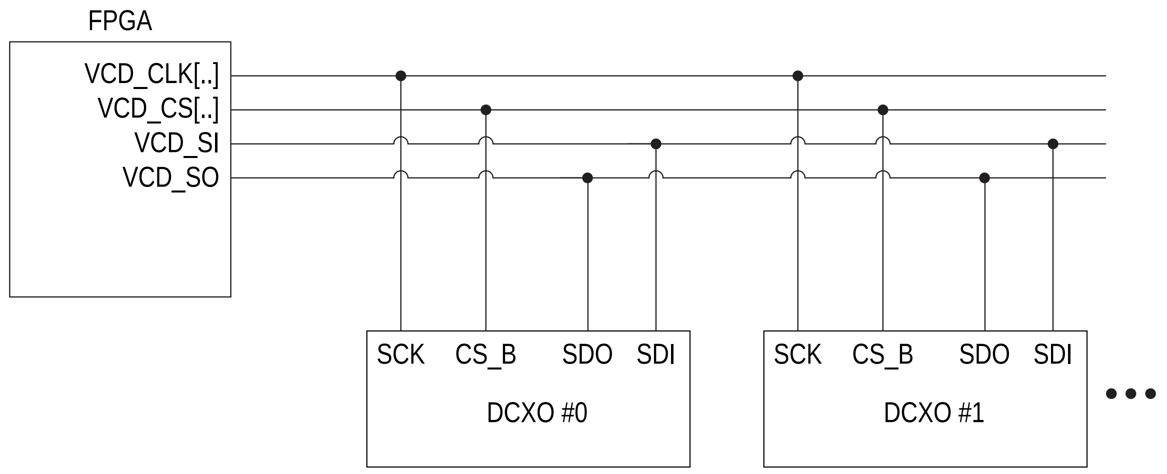
A DCXO places a number of requirements on the solution. You have to configure and reconfigure it according to the nominal clock frequency that you require. You might have to write to a large number of registers with values that you need to generate from a GUI-based tool from the DCXO manufacturer. You have to update it in real time with control values across a SPI bus. In a multi-port solution, several such devices can share the same bus.

The DCXO interface provides a simplified interface to the external DCXO and enables the real-time update transactions from the filter without a CPU intervening.

Overview

The DCXO interface block drives the SPI interface connected to the external DCXO device(s).

Figure 3: DCXO connections



For signal integrity reasons, the CS and CLK signals are point-to-point, while the SI and SO signals are point-to-multipoint.

Internally to the FPGA, the DCXO interface block receives input from two interfaces:

- The CPU bus for initial configuration. Once you set up a frequency, no more CPU intervention is required.
- The filter interface, which receives a constant stream of control values from the Filter.

The CPU interface has the same specification as the CPU interface on the Filter block and is described in more detail in the RTL deliverables.

The interface to the Filter is glueless and you just have to connect it correctly. The SPI interface supports multiple external devices. The DCXO block generates no interrupts.

Device Support

Currently, the DCXO interface only supports one external device type: the Microsemi ZL30240.

API

An application programming interface (API) provides a clean interface to the DCXO for the use of management software, and hides the low-level register interface both in the DCXO and in the FPGA.

Table 1: API Functions

Function	Description
Reset	Applies a reset to the DCXO internals.
Set Frequency	Applies the correct internal register settings to put the DCXO at a specified nominal frequency. The DCXO only supports preverified frequencies, as the optimal register configurations for a given frequency need to be generated manually.
Write Register	For debugging only. Allows direct writes to individual DCXO registers
Read Register	For debugging only. Allows direct reads from individual DCXO registers.

Support of new target frequencies can be added to the API on request.

The Filter

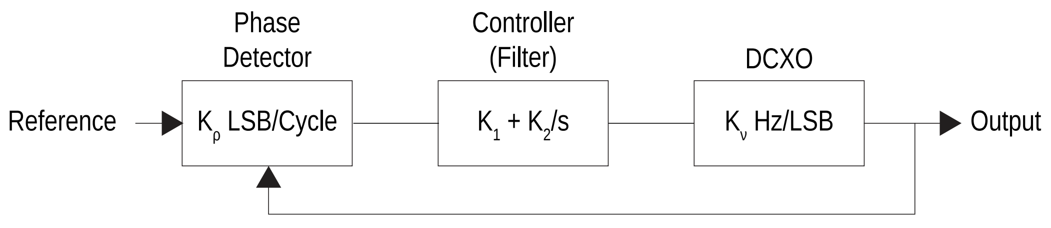
The filter performs the control operations that generate updates to the DCXO based on the difference between the incoming timing information and the actual TX reference clock that the FPGA generates. The mathematical functions are highly configurable to allow it to adapt to any surrounding environment.

Theory of Filter Operation

The transfer function of the loop controller (here termed the “filter”) defines the properties of PLL that the control loop forms, which typically takes the form of a type 2, second-order loop.

Figure 4: PLL Representation

The filter function is a PI controller, because it has a proportional (K_1) and an integral (K_2) part to its transfer function. The equations summarize the characteristics of the closed loop behavior in terms of the natural frequency ω_n (omega-n) and the damping factor ζ (zeta).

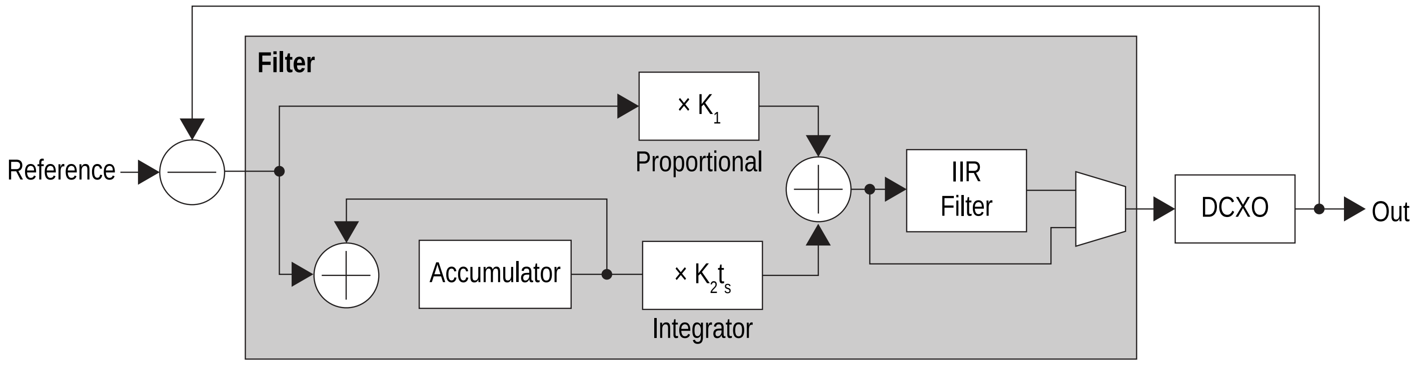


$$\omega_n = \mathrm{sqrt}(K_2 K_p K_v)$$
$$\xi = K_1 K_p K_v / (2\omega_n)$$

Filter Implementation

Figure 5: Filter Implementation

The filter internal blocks.



You can configure the block to support any number of individual ports. It performs its processing serially, in the order that the input ports are polled.

Filter Interfaces

Table 2: Filter Interfaces

Name	Description
CPU interface	Use for configuring and monitoring. The filter needs no real-time control once set up.
Input	Provides a phase difference signal, usually in units of bits. Supports a configurable number of ports.
Output	Provides a control value per port. Connects directly to the DCXO block.

The RTL for the filter includes detailed descriptions of these interfaces.

Internal Functions

The filter operates as a sampled system. When it receives a new input value, it processes it and produces an output value. Without any input values, the output does not change.

Note: All values that the filter processes are signed.

The filter provides a PI transfer function. The filter front-end adds each input sample to an accumulator (the integrator). Then, after it applies the proportional and integral gain factors, it adds the accumulator value to the input value.

Optionally, you can pass the values from the front-end through an IIR filter, typically to remove high frequency components arising from sampling noise. The IIR filter coefficients are fixed point 32-bit values, which you can use to program first-, second- or third-order filter functions.

After the IIR filter, the filter produces the samples on the output.

Table 3: Auxiliary Functions

The filter offers a number of auxiliary functions that you can configure.

Name	Description
Fast pull-in	You can configure the filter to move the integrator value faster than the actual input value can do. A configurable threshold allows you to apply a multiplier factor to the accumulator increment and decrements, when the input samples are greater than the threshold value.
Monitor	You can monitor a single port. Then the CPU has read access to all the internal states for that port. It can plot real-time response curves or monitor for values out of range, or just watch during system debugging.
Saturation	You can configure the saturation threshold to limit the maximum values of the integrator and the output.
Offsets	You can apply an offset to any input values.

API

The API provides a clean interface to the filter block and hides the individual registers from the user.

API Functions

Table 4: API Functions

Name	Description
Set IIR	Sets up one of a number of pre-programmed IIR filter options.
Set Filter	Sets the proportional and integral multiplication factors, saturation thresholds, pull-in functions, and offsets.
Monitor	Allows live monitoring of internal operation of the filter, for testing and debugging.

Using the API

To use the API:

1. Calculate the values.
2. Experiment with the values.
3. Adjust the values.

Filter Configurations

You derive the configuration for the filter based on the physical constants of the various components of the system, and what performance parameters you need to achieve.

System Constants

Table 5: Physical Constants

The physical constants that affect the filter configuration.

Physical Constant	Description
The units of the filter input, normally referred to as the phase detector constant, Kp.	Gives the gain of the phase differentiator (comparing the phase of the reference timing and the system output), typically in units of LSB/bit of the signal that you are controlling.
The output frequency change per unit filter output, or VCO constant, Kv.	Typically in units of bits/s/LSB. These bits/s (or Hz) refer to the same actual signal that the input units are referring to, which might not necessarily be the DCXO output clock but the data signal that this clock drives.
Sampling frequency.	Arbitrary, but should be at least a factor 10 above the frequency range of operation of the filter. Consider 6 kHz a minimum.

Given these constants, you can construct an equivalent PLL model of the system and calculate its expected behavior for varying values of the configuration parameters.

Primary Parameters

You must configure the loop to achieve the desired values for the following primary performance parameters, assuming a second-order system:

- 3dB bandwidth
- Damping factor ζ (zeta) or (equivalent) gain peaking limit

Depending on the signal you are reconstructing, the requirements might vary. The bandwidth can sometimes be specified, but more often has to comply with some maximum limits set by standards, and minimum limits set by the wander specifications for the network. Within these limits, typically set the bandwidth as high as possible while achieving the jitter attenuation necessary for compliance. Experiment with different values.

The limit on gain peaking applies in a network, such as SONET/SDH, where there might be many consecutive hops. The limit is: maximum 0.1dB gain peaking, corresponding to a damping factor ζ of approximately 4.5. Exceeding this limit might result in instability in a long daisy chain.

In the PI controller, the I (integrator) factor affects the damping factor. For any given P, a lower value of I gives a higher damping. Because of the high damping required, the value of I might be very small (less than 1) and is programmed in a floating-point format.

Jitter Compliance

Because of the PI controller model, the P (proportional) path allows a direct forwarding of noise at the input towards the output. Typically, the sampled nature of the timing information introduces noise to this path. The larger the P factor, the more noise is forwarded. You must reach a compromise with the other requirements on P. As P gets smaller, the damping factor (and therefore stability) reduces. The loop bandwidth is also reduced proportionally.

The IIR filter allows a further reduction in jitter, by providing a low-pass filtering of the output samples. However, to avoid the IIR filter response from affecting the loop stability, its cutoff frequency must be outside the pass-band of the loop. Altera recommends a factor of 6 between the loop bandwidth, and the IIR cutoff frequency.

Altera provides a number of standard IIR setups in the API, which are described by their order, and their cutoff frequency as a fraction of the sampling frequency (e.g. second order, Fs/500).

Calculations and Modeling

Altera provides several modeling templates:

- The SciLab script builds a model of the entire loop, including the IIR filter. This model provides output graphs of the overall loop transfer function, a zoomed view of the gain peaking area, and a step response in the time domain.
- An Excel spreadsheet, which calculates the performance parameters from the physical constants and the configuration parameters.

In addition, a SciLab script that performs IIR filter design (if you need a new filter setup) is also available.

SciLab is an open source program similar to MATLAB and is free to download and use.

Related Information

SciLab Website

Secondary Parameters

The most important secondary parameters are:

- Pull range
- Pull-in time

The DCXO constant (how much the frequency changes for a given filter output) gives the pull range, the integrator constant "I", and the effective size of the integrator. You can use the provided spreadsheet to calculate the constant. Adjust the integrator saturation value until the pull-in range is sufficient for the application, but not excessive. A very large saturation value can result in long startup times, as the integrator value might start at one of the extremes.

Pull-in time is related to the integrator saturation value. If the input signal is limited in amplitude, it may take a long time for the integrator to adjust completely to the target frequency.

Table 6: Parameters

Parameters that improve the pull-in time.

Parameter	
Limit Threshold	The input values must never exceed the Limit Threshold during normal operation. Exceeding this threshold indicates a signal interruption, and you should quickly adjust to a new target value for the integrator (which represents the ppm offset of the client signal).
Limit Add	This parameter varies how much faster than normal the integrator should adjust, when you exceed the Limit Threshold.

You set the parameters using the API. Use experimental measurements of how much input variation you see for each client.

PLL Model Validation

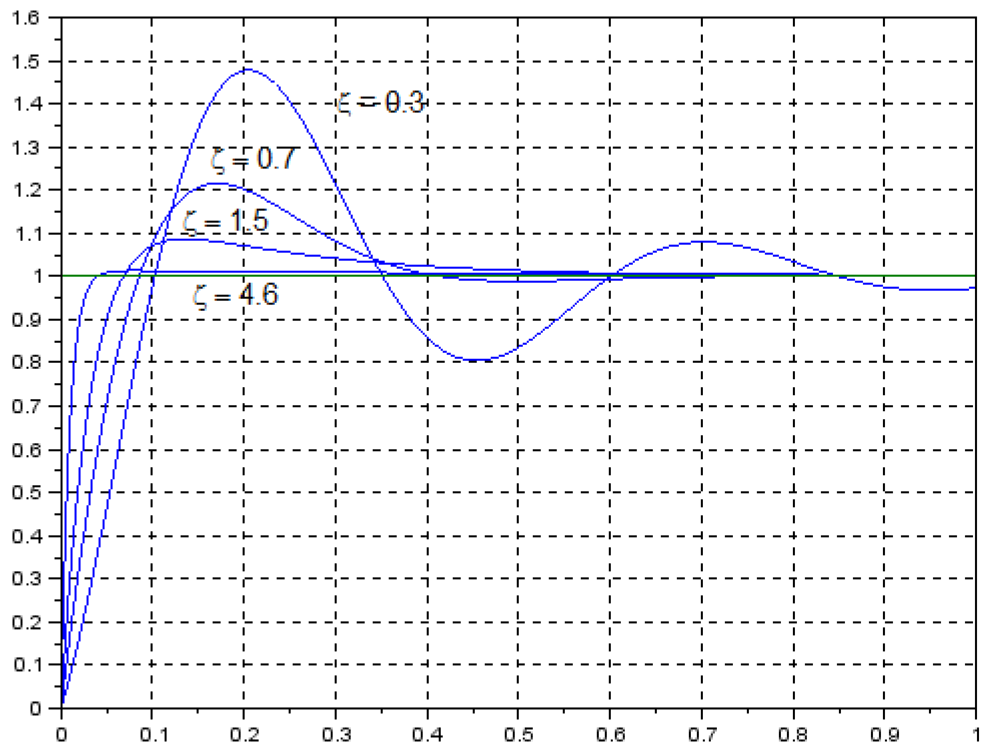
Assuming that the loop locks and a clock reconstructs successfully, you must validate that the FPGA calculated the system constants correctly and that the loop actually exhibits the dynamic behavior predicted by the modeling tools.

In simulations, the time constants can be of the order of hundreds of milliseconds. Additionally, the behavior of a very highly damped loop does not vary visibly at high damping factors. Thus, it might be difficult to see if the damping factor is for example, 3 (a fail) or 5 (a pass). Measuring the gain peaking can give an indication, but it is difficult to obtain an accurate measurement, and requires test equipment.

Altera recommends you validate this in the hardware, by applying an alternative configuration to the filter that gives it a much more visible second-order step response.

Figure 6: Step Response vs Damping Factor

The damping factors are (in decreasing order of damping): ζ = 4.6, 1.5, 0.7, 0.3.



You can estimate the damping factor, by examining how rapidly the peaks attenuate. Similarly, you can compare the natural frequency of the loop ω_n (again as the spreadsheet or SciLab model predicts) to the values you expect.

By taking a couple of datapoints with these underdamped configurations, you can determine if the loop behaves as per the model, or if one of the assumptions does not hold. For example, if the input values are in bytes, as opposed to bits.

If the loop behaves like the model in these configurations, you can be confident that the more highly damped setups also behave according to theory.

Example

The example describes the calculations for practical implementations of two clock reconstruction scenarios using an external Microsemi ZL30240 device as DCXO, and generating transmit reference clocks for:

- A 100G OTU4 interface
- A 4GFC interface

The following reference files for the example are in the **example_files** directory:

- **loop_parameters.xlsx**. Contains the calculations for the filter parameters
- **otu4_model.sce** and **4gfc_model.sce**. Each contains a SciLab model that demonstrates the dynamic response of the filter setup

Loop Parameter Spreadsheet

Table 7: Loop Parameter Spreadsheet

The **loop_parameters.xlsx** spreadsheet on the OTU4 tab has these elements.

Section	Color	Description
Top left	Light yellow fields	Fields that contain values for global constants for the system. For example, the reference clock for the ZL30240 DCXO is set to 48 MHz.
Client	Beige	Constants related to the client signal being controlled (e.g. the bitrate).
DCXO setup	Blue	Programming values for the DCXO in order to set up the nominal reference frequency.
Loop constants	Pink	Constants derived from the implementation and the choice of client type, DCXO setup, etc. Kp indicates that the units of the input to the filter from the phase detector are half-bits (i.e. that one cycle of client signal gives a result in 2 LSBs of input to the filter). Kv indicates the change in client signal rate resulting from the filter output, in this case the change in bits/s resulting from a one-LSB change in output value.
Filter configuration	Green	The actual values programmed into the filter registers.
Dynamic response	Purple	The characteristics of the dynamic response resulting from the given configuration, for example the loop bandwidth and the damping factor. The spreadsheet also calculates the ratio between the IIR filter cutoff and the loop bandwidth. These values should have a wide margin to avoid the IIR filter affecting the dynamic response.

Section	Color	Description
Pull range	Light blue	The pull range characteristics.
Misc	Salmon	Only the resulting sample rate for the DXCO is given here.

The derivation of Kv depends on the structure of the physical interface and how you configure the DCXO for the client. This example achieves the client bitrate of 111.8 Gbit/s through a 10-lane SERDES interface using a reference clock of 698.81 MHz, with a bitrate/refclock factor of 16. Using the API, you initially set up the DCXO with its Fbdiv and divider control values to generate this nominal frequency. The filter controls the actual rate live: the filter output updates the Fbdiv with a rate of one LSB on the output giving a 2^{-28} absolute change in Fbdiv.

The size of the integrator controls the effective pull range, through the **igtSat** field. This field affects the maximum deviation possible at the filter output, and hence how much the DCXO frequency can be pulled. In this example 98 ppm is possible, exceeding the required 50 ppm (derived from adding the required client tolerance (20 ppm), the tolerance of the local timing reference (10 ppm), and the accuracy of the DCXO crystal (20 ppm)).

Running the SCE File

Run the **otu4_model.sce** in SciLab to generate four graphs of the loop performance:

- IIR filter transfer function
- Overall loop transfer function
- Zoomed plot of gain peaking in loop transfer function
- Time domain step response plot

Figure 7: Gain Peaking Plot

The peak is approximately 0.09 dB, which corresponds to the value in the spreadsheet.

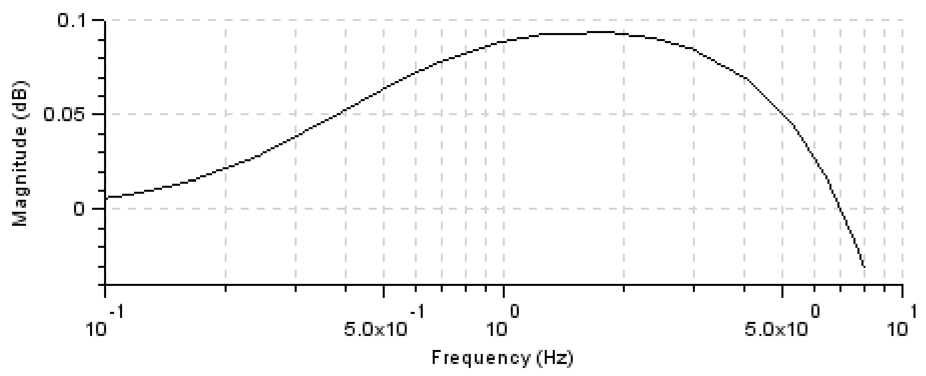
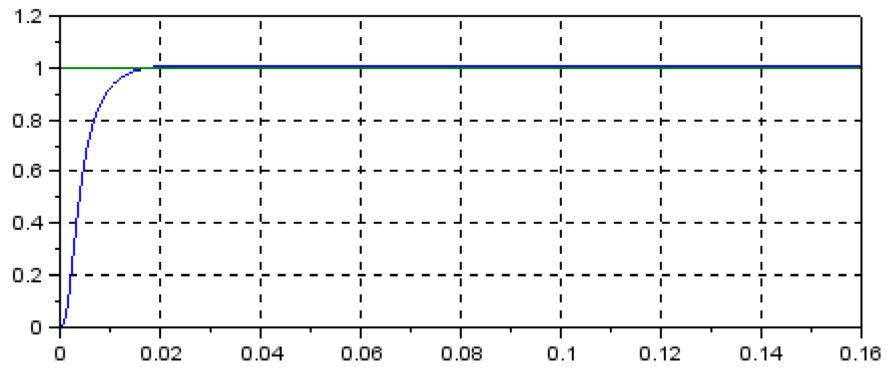


Figure 8: Time Domain Step Response Plot

A damping factor of 4.6 from the spreadsheet corresponds to a highly damped time domain response.



You can demonstrate similar results for the 4GFC interface, although based on different physical parameters.

Document Revision History

Date	Version	Changes
November 2013	2013.11.30	First published.