ClockMatrix™ - Channel Control for PTP with the Time of Day Counter

A PTP system needs to process the PTP packets and provide updates to the frequency, phase and time of day in the clock synthesis device via register access. In LinuxPTP for example, the ptp4l application serves as the PTP packet processor with a simple optional filter. The filter provides clock updates via the Linux PHC API to control a channel in ClockMatrix™. The PHC API has functions to adjust the frequency and phase of the channel. It also has functions to control the time of day (ToD) counter. The ClockMatrix clock synthesizer will then align its outputs to the 1Hz (or one pulse per second) rollover of the PTP ToD counter for use by the time stamper or other system blocks that need phase. Besides an explanation of how a PTP system works, this document also shows an example ClockMatrix configuration for use with SyncE and PTP.

Contents

1. Introduction

This application note explains how the PTP driver controls a channel in ClockMatrix by using the Time of Day (ToD) counter to align the device outputs to the 1Hz/1PPS (pulse per second) rollover of the counter. (The rollover of the counter is when the nanoseconds and sub-nanoseconds parts of the counter are zero.) The PTP driver also controls the clock used by the counter for both rough and fine adjustment of the output.

In a typical PTP system, there is a clock synthesizer channel with both frequency and phase control from the external software. This channel clocks a ToD counter. The counter value is incremented by the period of the channel clock on every edge. The outputs from the device use the ToD rollover clock for output alignment. In addition, the ToD counter can use different triggers to get values into and out of the counter including external 1Hz clocks, the internal rollover clock or a software command.

This application note describes how the PTP software will processes the packets, control the DPLL channel and control ToD counter. A separate program coordinates the ToD in the clock synthesizer and the time stamper. Both of these programs use the PHC functions to control of clocks and ToD in the different hardware devices.

In the last section of the document, there is ClockMatrix GUI configuration for an example PTP/SyncE system.

2. General Operation of a PTP System

To implement PTP, a system has four main tasks:

- Decode the incoming PTP packets to extract the signaling and time stamps.
- Create the sets of local time stamps and extracted time stamps from the incoming packets.
- A simple filter or complicated servo to transform the time stamp groups into clock updates.
- Finally, it needs an interface to send the clock updates to hardware.

The PHC API contains the code to translate the clock and ToD update functions into the register sequences specific for each device.

In an example Linux system running LinuxPTP in standalone mode, ptp4l will process the packets, extract the local and remote time stamps for processing, process the time stamps using its filter and send clock updates to the clock synchronizer via the PHC functions. When locking to a new master, the ptp4l software via the PHC API will adjust the ToD counter rollover to phase align all the output to the PTP 1 PPS from the master.

(The operation of Renesas PTP Clock Manager for Linux, pcm4l, is similar. The ptp4l program will still process the packets and create the time stamp groups as in standalone ptp4l. Then, the unfiltered time stamp groups go from ptp4l to pcm4l via a socket interface. Within pcm4l is an advanced non-linear adaptive servo to provide frequency and time transfer over more challenging networks. Like ptp4l, pcm4l will send the resulting updates to the clock synthesizer via the PHC API.)

When the system starts, the clock synthesizer will have an arbitrary frequency and phase before locking to a master. As ptp4l locked to the master, it will go through its internal states as defined in Table 1.

Table 1. Ptp4l State Definitions

Number	Name	Description
0	SERVO_UNLOCKED	The servo is not yet ready to track the master clock.
1	SERVO_JUMP	The servo is ready to track and requests a clock adjustment to correct the estimated offset.
2	SERVO_LOCKED	The servo is tracking the master clock.
3	SERVO_LOCKED_STABLE	The Servo has stabilized.

Note: These definitions are in servo.h in the ptp4l distribution.

First, ptp4l will wait for packets in the SERVO_UNLOCKED (S0) state.

Once ptp4l starts receiving packets, it will go to the SERVO_JUMP (S1) state. Ptp4l will estimate the phase offset between the master and the local clock. If it is larger than **first_step_threshold** in seconds, ptp4l will send an adjust time command to jump the ToD counter by the estimate of the offset via the PHC API. This will align the ToD counter to the local estimate of the master 1-PPS phase. Based on the internal output from the ToD counter, the device will align its output clocks to the new 1-PPS alignment of the ToD counter on the next ToD rollover. Finally, ptp4l will start collecting time stamp groups and make a new estimate of the offset.

(As of LinuxPTP version 3.2, ptp4l will discard packets after the phase adjustment. The delay to allow the time stamper to match to the new ToD alignment. The **step_window** parameter defines the number of packets to discard before continuing.)

If the offset is still larger than the **step_threshold** parameter in units of seconds in ptp4l.cfg, ptp4l will perform another phase adjustment. Typically, both **first_step_threshold** and **step_threshold** are set to 20 microseconds. (The **step_window, first_step_threshold** and **step_threshold parameters** are in the ptp4l configuration file.)

Once the estimated offset is small enough, ptp4l will switch to the SERVO_LOCKED (S2) state and the filter will switch to frequency offsets via the PHC API to pull-in the remaining phase offset between the local ToD and the estimate of the master ToD. For LinuxPTP version 3.0 or newer, the servo will switch to the SERVO_LOCKED_STABLE (S3) state when the offset is small. In the S3 state, the filter will switch to phase offsets via the PHC API to get the best alignment with the master. To get to the S3 state, the last **servo_num_offset_values** values of the locally estimated offset to the master are less than **servo_offset_threshold** as defined in the ptp4l configuration file. For ClockMatrix, the recommended value for **servo_offset_threshold** is 100 in units of nanoseconds and the value for **servo_num_offset_values** is 64 in units of packets**. (**The **servo_offset_threshold** and the **servo_num_offset_values** parameters are also in the ptp4l configuration file.)

3. Aligning the Clock Synthesizer to the Time Stamper

As the clock synthesizer aligns the ToD to the master, the system also needs to align the time stamper to the clock synthesizer. The clock synthesizer typically sends two physical clocks to the time stamper: a PTP clock (high frequency usually 250MHz) and a PTP time stamp event (1Hz). A system can use other low frequency clocks from 0.5Hz to 4kHz as long as the time stamper can identify the clock edge corresponding to the rollover event. The time stamper uses the frequency from the PTP clock to increment its ToD counter. It uses the low frequency clock to latch a time stamp to coordinate between the ToD in the clock synthesizer and the time stamper.

A PTP system needs a method to align the ToD in the time stamper with the ToD in the clock synthesizer. In LinuxPTP, the ts2phc program coordinates between the clock synthesizer and time stampers when they are in separate devices. Under the control of ts2phc, the time stamper latches its ToD at the instant when the edge of the external 1PPS input arrives from the clock synthesizer. Ts2phc requests this ToD value via the PHC API to determine the difference between the seconds on the clock synthesizer ToD and the value from the time stamper and requests a ToD adjustment on the time stamper via the PHC API. Depending on the time stamper, the resulting ToD alignment can be less than 100 ns by accurately sampling the 1PPS clock and assuming that the nanoseconds and sub-nanoseconds part of the arrival of the 1Hz clock edge is zero. After the phase adjustment, the clock synthesizer changes the time stamper ToD phase via small frequency or phase changes via the high-speed PTP clock.

4. Clock Control using the Linux PTP Hardware Clock (PHC) API

The PTP application uses the PTP Hardware Clock (PHC) subsystem to request frequency, phase and ToD changes to the clock.

In LinuxPTP, each clock synthesizer used for PTP has its own PHC interface. The purpose of the PHC APIs is to translate the control commands from the PTP software to specific register sequences for each device. The Linux PHC drivers for ClockMatrix are part of the kernel.

A typical PHC driver supports the functions in Table 2.

Table 2. PHC API Functions

Name	Description
getTime	Gets to the ToD counter value (immediate).
setTime	Set the ToD counter to a specific value (immediate).
adjTime	Make a relative change to the ToD counter.
adjFine	Change the frequency offset to the specified value (resolution is ppb). This function replaced the adjFreq in earlier version of the PHC interface.
adjPhase	Using the write phase mode in ClockMatrix, adjust the phase by the specified value.
EXTTS	Latch the ToD counter on the edge of an external signal (usually 1 PPS) where EXTTS is an EXTernal Time Stamp event.
PEROUT	Control the periodic outputs on the device (used to control the 1 PPS output from the clock generator to the time stamper for frequency/phase distribution in a system).

The hardware (clock synthesizer or time stamper) processes the getTime and setTime functions when the command arrives. There may be variable delays in accessing the hardware depending on the system and the serial bus loading. As a result, the system does not use these commands for fine control of the ToD. The adjTime command is very precise since it is relative to the last ToD value and any application delays do not impair its accuracy. While PTP software can use either adjFreq or adjFine to adjust the frequency of the device, the adjFine is preferred due to higher accuracy updates.

For ClockMatrix specifically, PHC API functions have an “idtcm” prefix. The idtcm_settime PHC command will set the ToD in the clock synthesizer to an arbitrary value using an immediate write while the idtcm_adjtime PHC command will set the ToD to a relative value using a relative write. Both are expected step the ToD and clock synthesizer will align the output signals to the new ToD rollover phase after the ToD is changed.

The ClockMatrix PHC driver switches the channel mode between “write frequency” for idtcm_adjfine and “write phase” for idtcm_adjphase modes as needed. For the idtcm_adjfine and idtcm_adjphase PHC commands, the phase change relative to the previous frequency or phase of the output, so the outputs maintain their alignment to the ToD counter rollover.

For all PHC functions, the PHC driver and the ClockMatrix hardware handles the alignment of the outputs without additional effort from the higher-level application.

5. Example GUI Configuration for SyncE/PTP on 8A34001

For this example, the SyncE is on channel 1 and the PTP is on channel 2. The SyncE recovered input is on clk4 and the SyncE transmit clock output is on Q3. The PTP Clock is on Q4. The PTP 1PPS is on Q5. For this configuration, the ToD counter and ToD counter output alignment are enabled for the PTP channel. (Other channels with PTP outputs would need frequency from the PTP channel and optionally alignment from the PTP channel's ToD counter.)

The PHC driver uses a .bin file for configuration. The .bin file must be in the target file system usually in the /lib/firmware directory. The .bin file has two pieces: the driver configuration and an optional set of device configuration registers. The driver configuration contains the channel, ToD and output list. If the registers are in the bin file, the driver resets the device after loading the registers and the loading process may interrupt the output clocks.

For this example, the GUI would generate a .bin file with a driver configuration and a full set of register values.

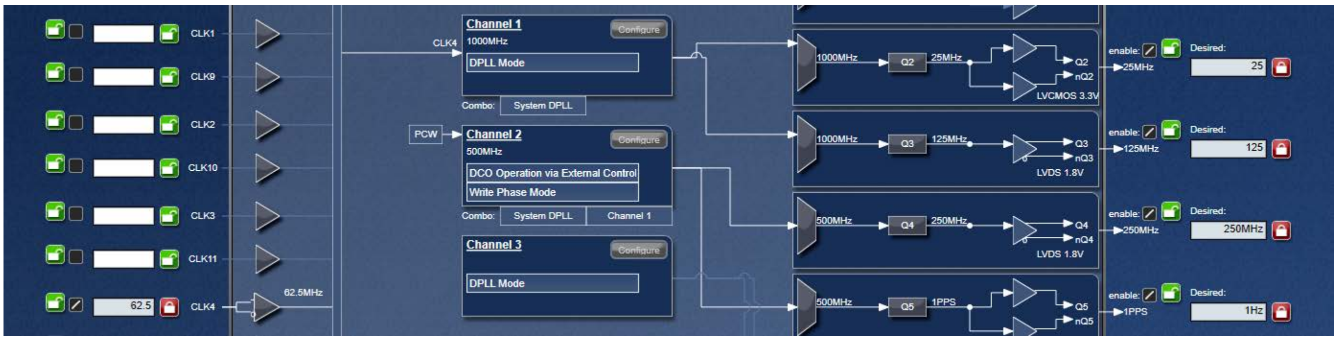
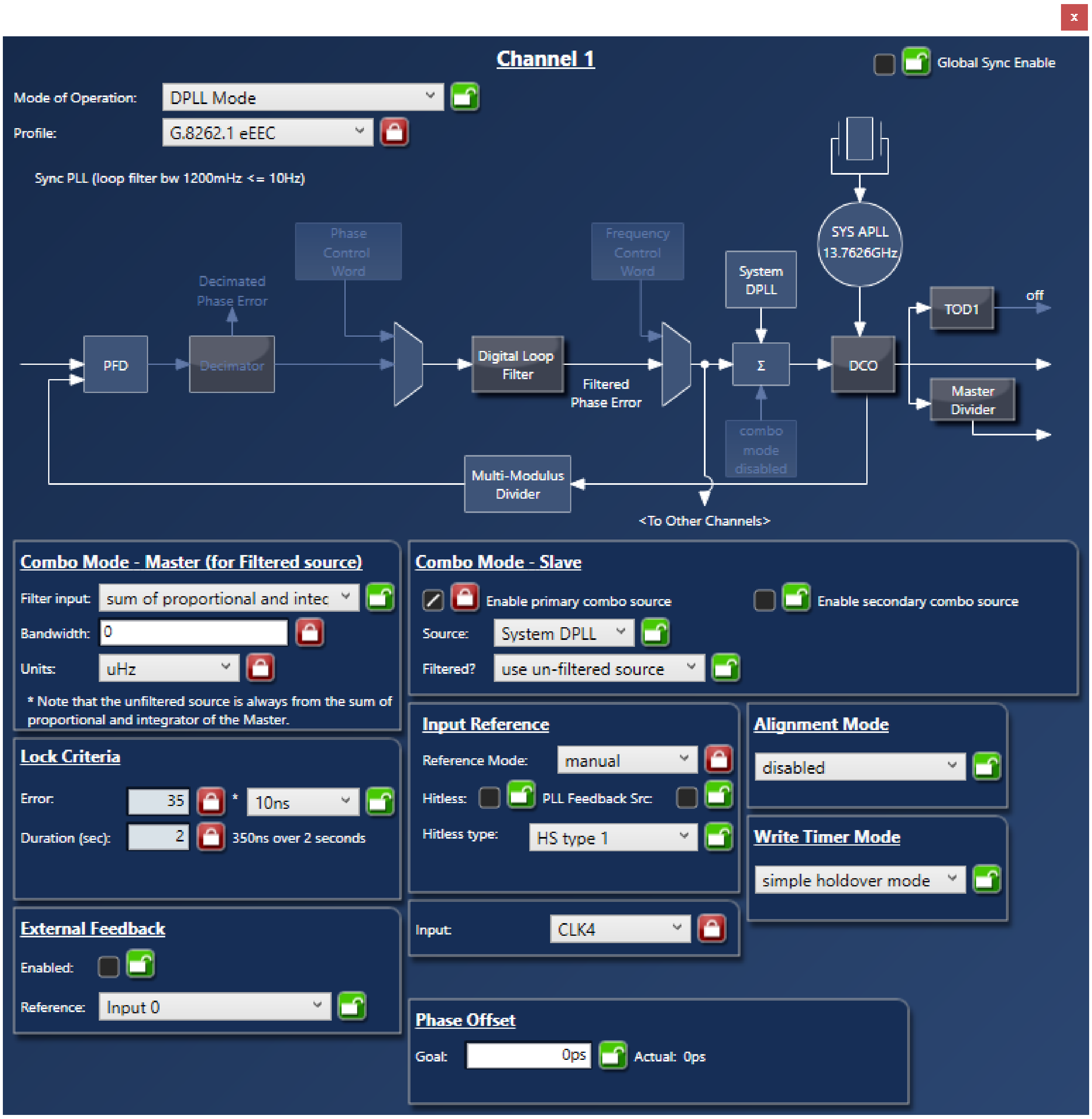


Figure 1. Overall Configuration



Mode of Operation: DPLL Mode
Profile: G.8262.1 eEEC
Sync PLL (loop filter bw 1200mHz <= 10Hz)

Combo Mode - Master (for Filtered source)
Filter input: sum of proportional and intec
Bandwidth: 0
Units: uHz
* Note that the unfiltered source is always from the sum of proportional and integrator of the Master.

Combo Mode - Slave
Enable primary combo source
Enable secondary combo source
Source: System DPLL
Filtered? use un-filtered source

Lock Criteria
Error: 35 * 10ns
Duration (sec): 2 350ns over 2 seconds

Input Reference
Reference Mode: manual
Hitless: PLL Feedback Src:
Hitless type: HS type 1

Alignment Mode
disabled

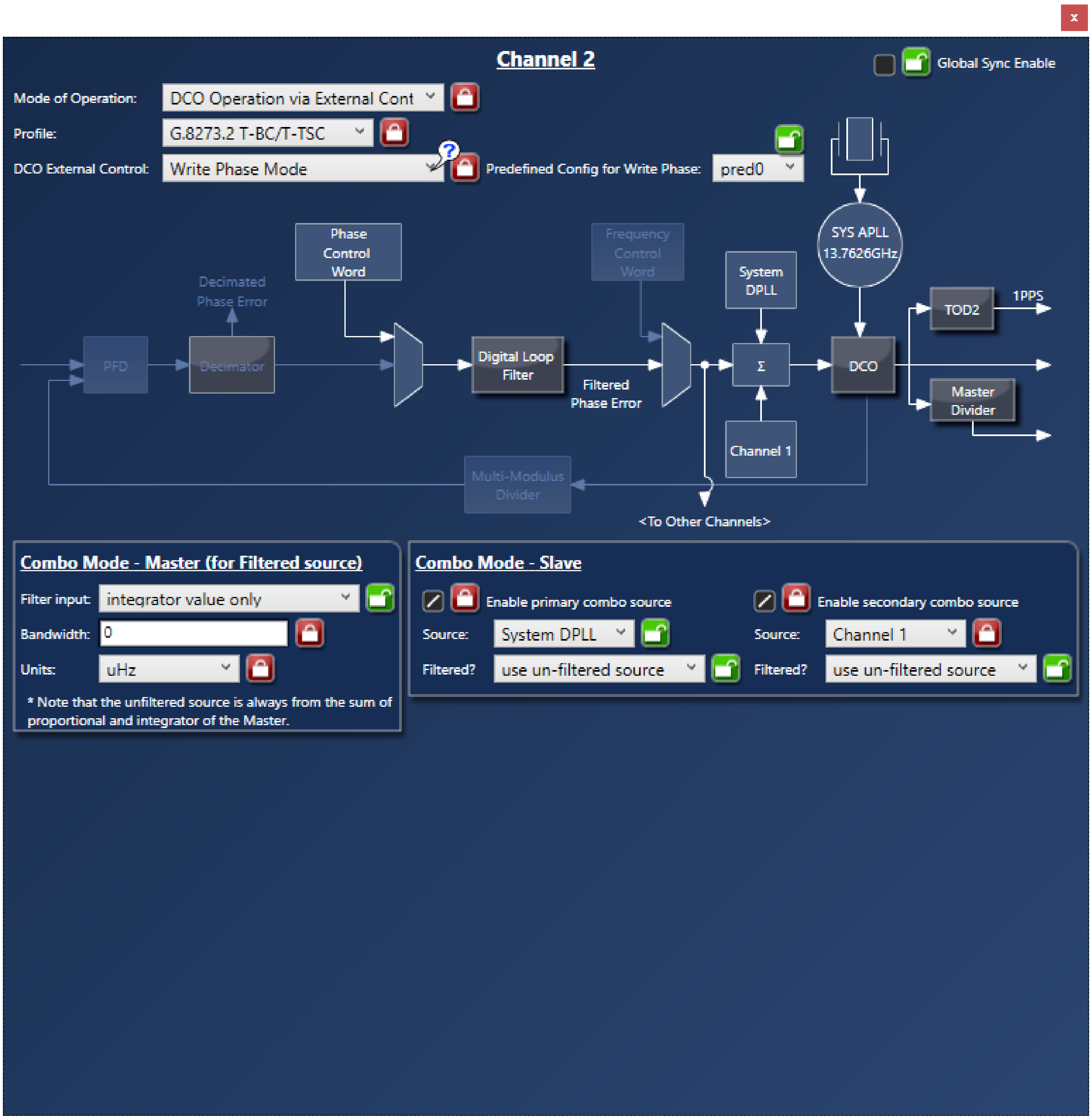
Write Timer Mode
simple holdover mode

External Feedback
Enabled:
Reference: Input 0

Input: CLK4

Phase Offset
Goal: 0ps Actual: 0ps

Figure 2. SyncE (DPLL) Channel Configuration



Mode of Operation: DCO Operation via External Cont

Profile: G.8273.2 T-BC/T-TSC

DCO External Control: Write Phase Mode

Predefined Config for Write Phase: pred0

Combo Mode - Master (for Filtered source)

Filter input: integrator value only

Bandwidth: 0

Units: uHz

* Note that the unfiltered source is always from the sum of proportional and integrator of the Master.

Combo Mode - Slave

Enable primary combo source

Source: System DPLL

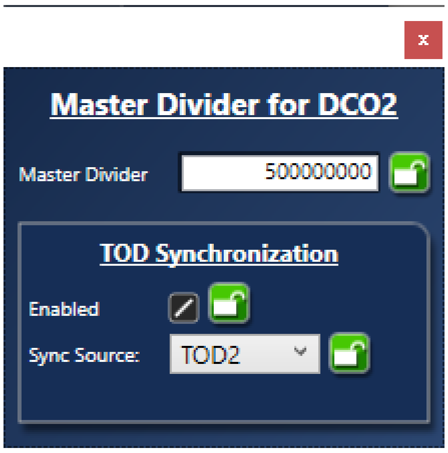
Filtered? use un-filtered source

Enable secondary combo source

Source: Channel 1

Filtered? use un-filtered source

Figure 3. PTP (DCO) Channel Configuration



Master Divider for DCO2

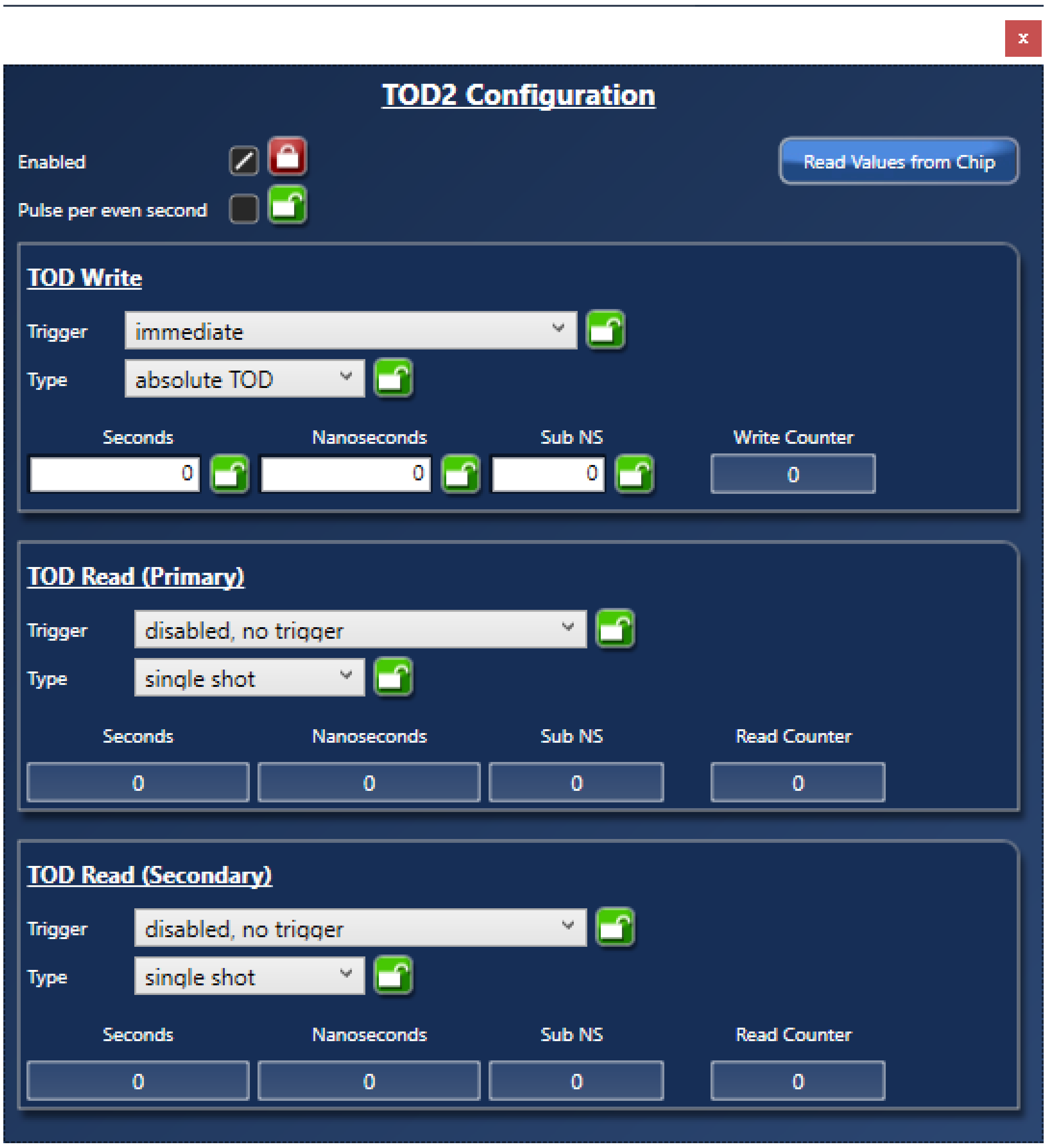
Master Divider: 500000000

TOD Synchronization

Enabled: ✓

Sync Source: TOD2

Figure 4. PTP Master Divider Configuration



TOD2 Configuration

Enabled: ✓

Pulse per even second: ☐

TOD Write

Trigger: immediate

Type: absolute TOD

Seconds	Nanoseconds	Sub NS	Write Counter
0	0	0	0

TOD Read (Primary)

Trigger: disabled, no trigger

Type: single shot

Seconds	Nanoseconds	Sub NS	Read Counter
0	0	0	0

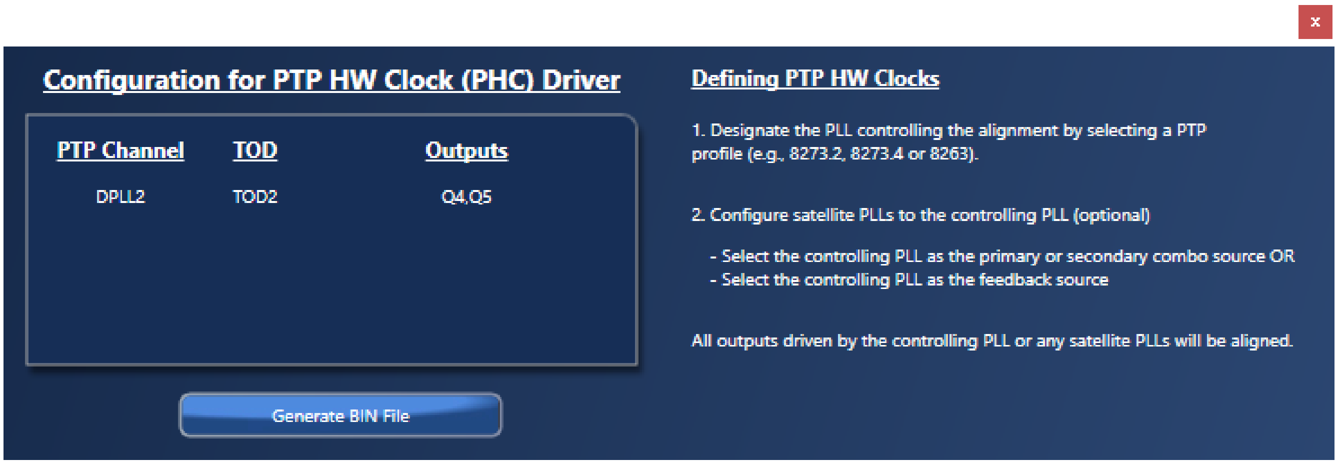
TOD Read (Secondary)

Trigger: disabled, no trigger

Type: single shot

Seconds	Nanoseconds	Sub NS	Read Counter
0	0	0	0

Figure 5. PTP ToD0 Configuration



Configuration for PTP HW Clock (PHC) Driver

PTP Channel	TOD	Outputs
DPLL2	TOD2	Q4,Q5

Defining PTP HW Clocks

1. Designate the PLL controlling the alignment by selecting a PTP profile (e.g., 8273.2, 8273.4 or 8263).

2. Configure satellite PLLs to the controlling PLL (optional)

- Select the controlling PLL as the primary or secondary combo source OR
- Select the controlling PLL as the feedback source

All outputs driven by the controlling PLL or any satellite PLLs will be aligned.

Figure 6. Generate Configuration for PHC Driver (.bin file)

6. Revision History

Revision	Date	Description
1.0	Jun 11, 2021	Initial release

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.