ClockMatrix™ Oscillator Compensation

This application note explains how to implement oscillator compensation in ClockMatrix to improve the holdover performance of a system. It introduces different methods for predicting how oscillators change frequency as the system runs. It also explains the programming of the registers needed to compensate the oscillator frequency for the entire device or for each channel separately.

Contents

1. Introduction

The system uses the frequency stability of the oscillator to maintain phase stability. During holdover, some PTP applications are required to maintain 1µs over 24 hours versus the locked phase, while a high quality uncompensated OCXO will only maintain 1µs for about 1 hour. A system can meet these stricter requirements by using an algorithm running on an external CPU to model the oscillator frequency variation versus time and temperature and write these corrections to the ClockMatrix device. These external algorithms allow the system to meet the phase holdover specification.

An oscillator manufacturer describes its products with different frequency specifications: offset (calibration), aging (drift), temperature stability (frequency versus temperature) and stability versus other parameters (like voltage or load). The offset (or calibration) is the frequency difference between the output and its nominal frequency. A PLL can remove a stable offset versus an external reference. The aging (or drift) is a variation of the offset over time in units of frequency change per day or frequency change per year. The temperature stability will depend on the type of oscillator. For an OCXO, the use of insulation around the crystal oscillator with a heating element in the package will keep the crystal at a constant temperature. This minimizes the impact of temperature on frequency variation so the drift is the more significant factor. For a TCXO, temperature effects are more significant. The oscillator manufacturer should be able to provide a model of frequency change versus temperature for their oscillators. In addition, an oscillator will have frequency variation versus voltage and load, but a telecom system should be able to control for these effects via voltage regulators and buffering the clock signal.

2. Available Techniques for Holdover Accuracy

For an uncompensated oscillator, the DPLL will provide holdover filtering in the DPLL channel. While it is good for some applications, it will not be sufficient since it only measures the average long-term historical frequency of the oscillator versus the reference. Any future change in the oscillator frequency will cause a similar error in the holdover performance.

To improve the holdover performance, the system must predict the future changes in the oscillator frequency. There are two different approaches to the modeling of frequency variation: learning the performance based on local measurements of the oscillator versus a system input or using model parameters measured for each oscillator during manufacturing.

2.1 OCXO/TCXO Learning for Long-Term Time Holdover

For this method, the system will implement a standard model in external software to vary frequency over time. This algorithm could be common for different models and oscillator manufacturers. The model would have to learn the aging of the oscillator. Optionally, it would also have to model the effects of temperature by comparing the measured frequency versus an external temperature sensor near the oscillator accessible via an I2C interface. The customer or oscillator vendor must provide the correction algorithm. The algorithm must also provide controlled, precise adjustments to minimize wander transfer to the outputs of the DPLL channels.

The system calculates the model parameters when locked to a reference, but needs to apply these adjustments when in locked and holdover states to provide better long-term time holdover performance. A typical system would use SyncE or GNSS (GPS) derived reference to provide the nominal frequency for accurate modeling.

As shown in Figure 1, there are two parts to this method: the compensation algorithm running on the external CPU and the compensation of the OCXO frequency input which is within the ClockMatrix device.

A simple learning algorithm would measure the offset and drift of the oscillator versus a good external reference while the external reference is available. It would continue to make adjustments to keep the frequency of the oscillator plus the adjustment stable when the reference was removed and the system goes to holdover.

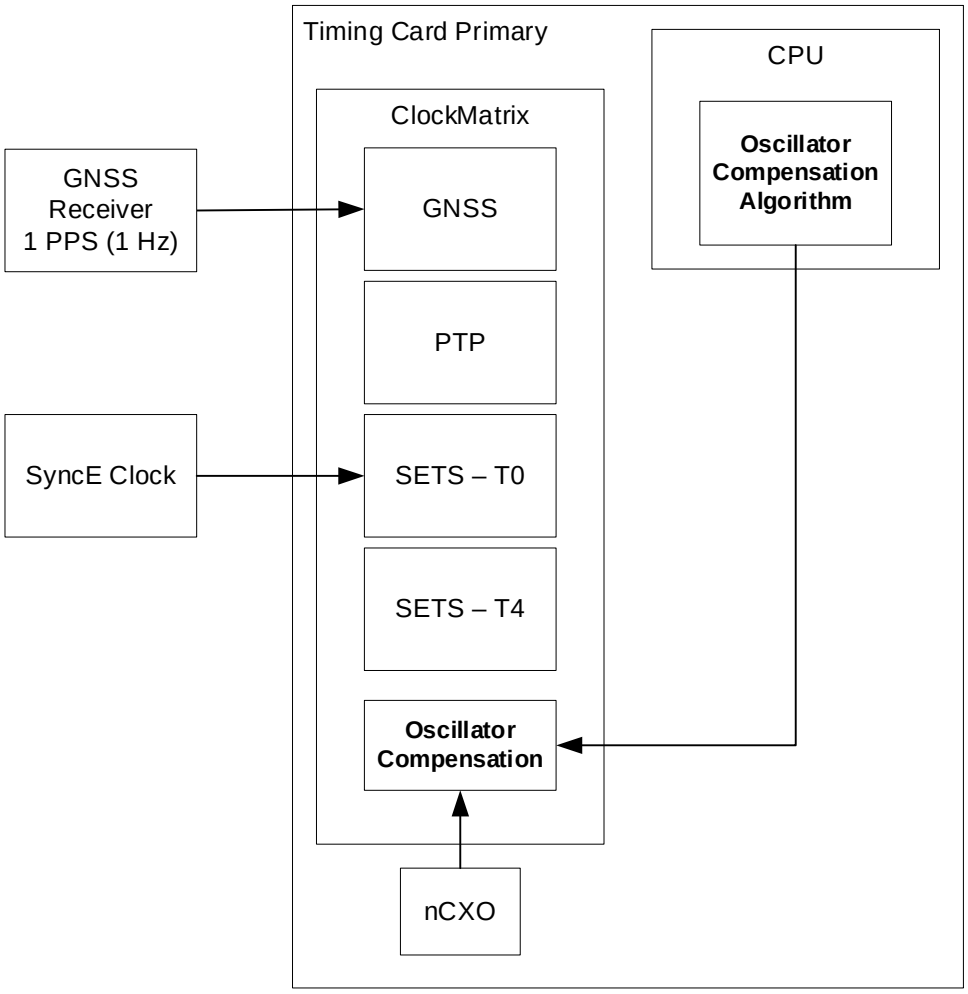

Figure 1. General Aging Compensation for Long-Term Time Holdover

2.2 Smart-OCXO Compensation for Long-Term Time Holdover

A second method for frequency adjustments uses a temperature sensor and frequency modeling parameters from a smart-OCXO. The smart-OCXO contains parameters for a frequency variation model supplied by the manufacturer. The manufacturer-supplied model will vary the frequency with temperature and time. The manufacturer will individually characterize each oscillator in the factory. The vendor will characterize *each* smart-OCXO over temperature before shipping, then it stores the values in an internal smart-OCXO EEPROM accessible via an I2C interface. More information about the models used are available from each vendor.

Similar to the previous method, the algorithm automatically compensates a DPLL channel via controlled, precise adjustments to minimize noise transfer to the outputs. The adjustments are applied independent of the DPLL's state but will have the most impact when the DPLL is in holdover. Unlike the previous section, this method does not depend on a local reference.

As shown in Figure 2, the system uses the temperature sensor and the parameters from the smart oscillator as the external input for the model.

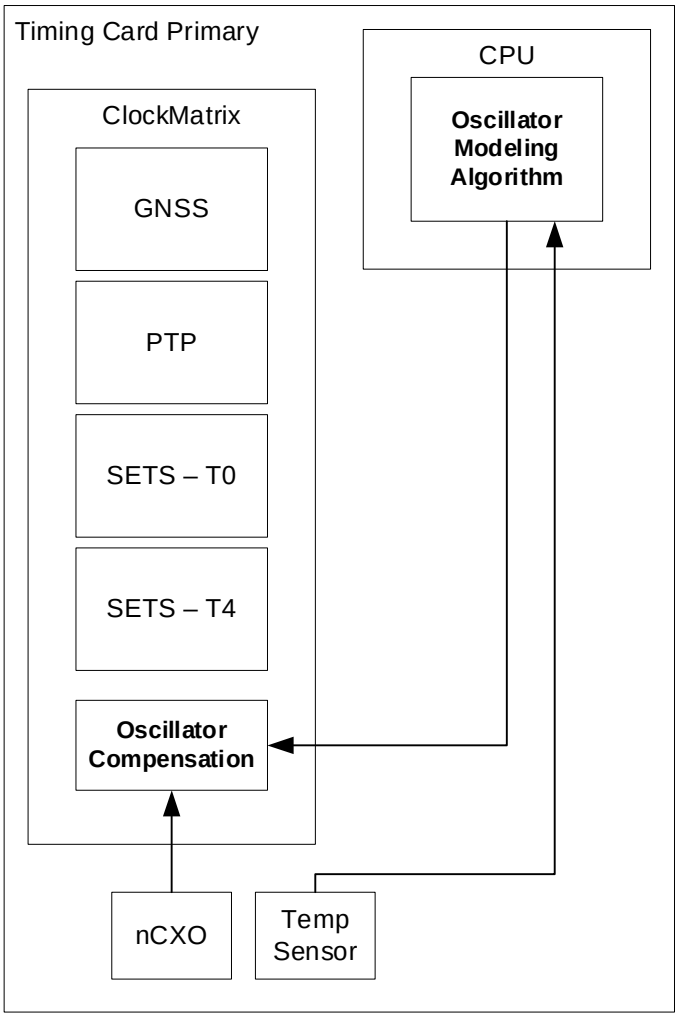

Figure 2. OCXO/TCXO Learning for Long-Term Time Holdover

3. Implementing per channel CPU Based Techniques with ClockMatrix

For implementing the frequency adjustments on ClockMatrix the DPLL_CTRL_0.DPLL_COMBO_SW_VALUE_CNFG register can be used. This register needs to be set for each channel by the external software. It is used in conjunction with the System DPLL combo bus feature when System DPLL is locked to external TCXO or OCXO. The value of this register is included in the summation that includes the output of the digital loop filter (or frequency write), the primary combo bus value and the secondary combo bus value. The summed frequency offsets control DCO frequency. Figure 3 shows the channel block diagram from the GUI and the location where the firmware adds this value.

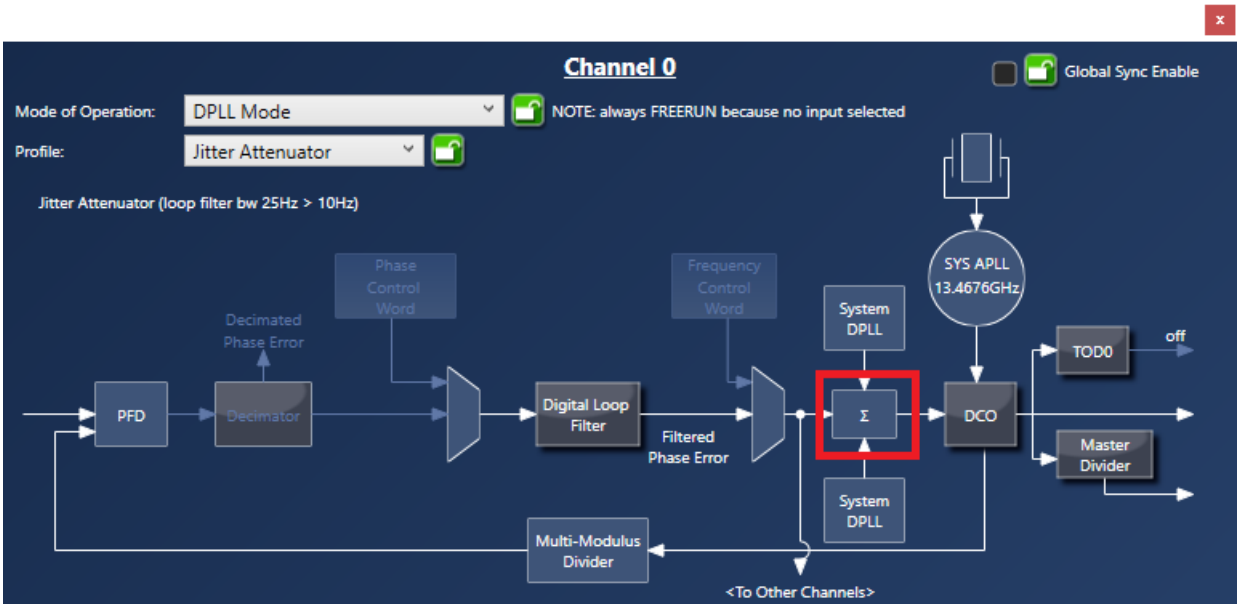

Figure 3. Location of Sum for DPLL_COMBO_SW_VALUE_CNFG Register in Channel

For this register, each channel would have the System DPLL as a combo mode source, and the DPLL_CTRL_{N}.DPLL_COMBO_SW_VALUE_CNFG register would have to be set for each channel as the adjustment.

The offset range is ±15625ppm while the range of a DPLL or DCO channel is ±244ppm. The value is stored as a signed 48-bit FFO in units of 2 ^ (-53).

Figure 4 shows the exact conversion between an offset and the value for this register. This equation is from the ClockMatrix datasheet in the “Write-Frequency” Mode section. The approximate conversion is:
FCW = (FFO/1E6)/(2^-53).

The formula for calculation of the FCW from the fractional frequency offset (FFO) is:

$$FCW = \left(1 - \frac{1}{\left(1 + \frac{FFO}{10^6}\right)}\right) \times 2^{53}$$

Where,
FFO = Fractional Frequency Offset, in ppm
FCW = Frequency Control Word (Positive or Negative Integer)

Figure 4. Conversion of FFO to FCW Register Value

When programming the DPLL_COMBO_SW_VALUE_CNFG register, the value will change the frequency after the highest address register byte is written to this register since every register in DPLL_CTRL_{N} module is a trigger register. Figure 5 shows the definition of the DPLL_COMBO_SW_VALUE_CNFG register from the device Programming Guide.

DPLL_CTRL_0.DPLL_COMBO_SW_VALUE_CNFG

DCO value to be added to the combo bus in SW combo mode.

Table 271: DPLL_CTRL_0.DPLL_COMBO_SW_VALUE_CNFG Bit Field Locations and Descriptions

Offset Address (Hex)	DPLL_CTRL_0.DPLL_COMBO_SW_VALUE_CNFG Bit Field Locations							
	D7	D6	D5	D4	D3	D2	D1	D0
028h	DPLL_COMBO_SW_VALUE_CNFG[7:0]							
029h	DPLL_COMBO_SW_VALUE_CNFG[15:8]							
02Ah	DPLL_COMBO_SW_VALUE_CNFG[23:16]							
02Bh	DPLL_COMBO_SW_VALUE_CNFG[31:24]							
02Ch	DPLL_COMBO_SW_VALUE_CNFG[39:32]							
02Dh	DPLL_COMBO_SW_VALUE_CNFG[47:40]							

DPLL_CTRL_0.DPLL_COMBO_SW_VALUE_CNFG Bit Field Descriptions			
Bit Field Name	Field Type	Default Value	Description
DPLL_COMBO_SW_VALUE_CNFG[47:0]	R/W	0	DPLL Combo SW value in units of 2^(-53).

Figure 5. Definition of DPLL_COMBO_SW_VALUE_CNFG Register

3.1 Example Calculation

This section shows a sample conversion of a ppm value to the 48-bit signed value needed for the DPLL_COMBO_SW_VALUE_CNFG register.

3.1.1. Example 1 - Correct by -3.5ppm

-3.5ppm = -3.5E-6

Using the approximate equation:

Value = int(-3.5E-6/(2^(-53))) = -31,525,197,391

Register = 2^48 -31,525,197,391 (since negative)

= 281443451513264 => 0xFFF8 A8F3 A9B1

Using the exact equation:

-3.5ppm => 0xfff8 a8f1 faae

3.1.2. Example 2 - Correct by +3.5ppm

3.5ppm = 3.5E-6

Using the approximate equation:

Value = int(3.5E-6/2^(-53)) = 31,525,197,392

Register = 31,525,197,392 => 0x0007 570c 564f

Using the exact equation:

3.5ppm => 0x0007 570a a74e

Renesas Application Engineering can provide example Python scripts to implement the conversion for reference.

4. Implementing SysDPLL based Offset with CPU Based Techniques

For this method, the user adjusts the SysDPLL frequency directly and the other channels use the SysDPLL frequency via the combo bus. The advantage of this method is that the SysDPLL frequency is used for input reference monitoring (including the fractional frequency offset measurement) and is used by the DPLLs as a reference for all calculations including fast lock and other internal processes.

Normally, the other DPLLs will filter any frequency change on the SysDPLL output. The individual adjustments still need to be small to prevent the other DPLLs from unlocking or increasing the wander generation on the outputs.

The simplified procedure to implement this technique is:

1. Get offset between XO_DPLL and “good” reference from SyncE or GNSS (GPS) receiver.
2. Get temperature from external sensor (if needed for model).
3. Process the measured offset (and temperature coefficient model) for oscillator to get FFO adjustment (a more complicated algorithm will limit the frequency step in a single change, but get the correct offset over time via a software frequency change limit).
4. Set SysDPLL to holdover.
5. Change nominal SysDPLL reference frequency to account for calculate offset from model.
6. Set SysDPLL to original mode (locked to XO_DPLL or reference input pin).

4.1 Pseudocode Implementation of Technique

When implementing this code, the holdover time should be as short as possible since the crystal accuracy will define the accuracy of the SysDPLL output during holdover. Since the SysDPLL bandwidth is wide, it will relock quickly to the OCXO once the adjustment is set. For a real system, the system should use a more complicated algorithm instead of using the offset between the “good” reference and the OCXO input shown in this section. The most important part of this example is the sequence of registers reads and writes for the ClockMatrix device.

```
goodRef #reference used for measurement of XO from SyncE or GNSS

oldMode = Read SYS_DPLL.SYS_DPLL_MODE.STATE_MODE

curRef = Read SYS_DPLL.SYS_DPLL_REF_PRIORITY_1

goodRefFFOValue = Read  STATUS.IN{goodRef}_MON_FREQ_STATUS.FFO

goodRefFFOUnits  = Read Get STATUS.IN{goodRef}_MON_FREQ_STATUS.FFOUNIT

goodRefFFO = goodRefFFOValue * goodRefFFOUnits #the FFO measurement is relative to the
XO_DPLL input

If curRef = 0x12 (XO_DPLL)

       #frequency of XO_DPLL input is M/N in Hz

       xoFreqM = Read  SYS_DPLL_XO.XO_FREQ.M

       xoFreqN = Read  SYS_DPLL_XO.XO_FREQ.N

       xoFreqM = xoFreqM * (1 – goodRefFFO) #replace with customer algorithm

       Write SYS_DPLL.SYS_DPLL_MODE.STATE_MODE to 3 (force holdover)

       Write SYS_DPLL_XO.XO_FREQ.M to xoFreqM

       Write SYS_DPLL.SYS_DPLL_MODE.STATE_MODE to oldMode
```

```
Else
       #curRef is refX
       #frequency of XO_DPLL input is M/N in Hz
       xoFreqM = Read INPUT_{curRef}.IN_FREQ.M
       xoFreqN = Read INPUT_{curRef}.IN_FREQ.N
       xoFreqM = xoFreqM * (1 – goodRefFFO) #uses instantaneous frequency for this simple
example
       Write INPUT_{curRef}.IN_FREQ.M to xoFreqM
       Write SYS_DPLL.SYS_DPLL_MODE.STATE_MODE to 3 (force holdover)
       Rewrite INPUT_{curRef}.MODE #trigger register for INPUT_{curRef] module
       Set SYS_DPLL.SYS_DPLL_MODE.STATE_MODE as oldMode #keep holdover as short as possible
```

5. Example Bench Test

To test the frequency-learning algorithm in Section 2.1, the bench would need a “good” input to measure the drift of OCXO. In a real system, this would come from a GNSS receiver. For a bench test, this would come from a function generator locked to the lab reference. Once the system is locked to this reference for a sufficient amount of time to allow the algorithm used to learn the oscillator parameters, the user puts the system into holdover.

In Figure 6, the output of importance is from the PTP channel. Normally, this would be 1Hz (1 PPS) with a tight phase holdover requirement. The GNSS input reference block in the ClockMatrix device measures the frequency difference between the frequency of 1-PPS input clock and OCXO input. The figure shows PTP (DCO) channel as the channel for measurement but the test can also be run with a SyncE (DPLL) channel or GNSS (up-convert) channel.

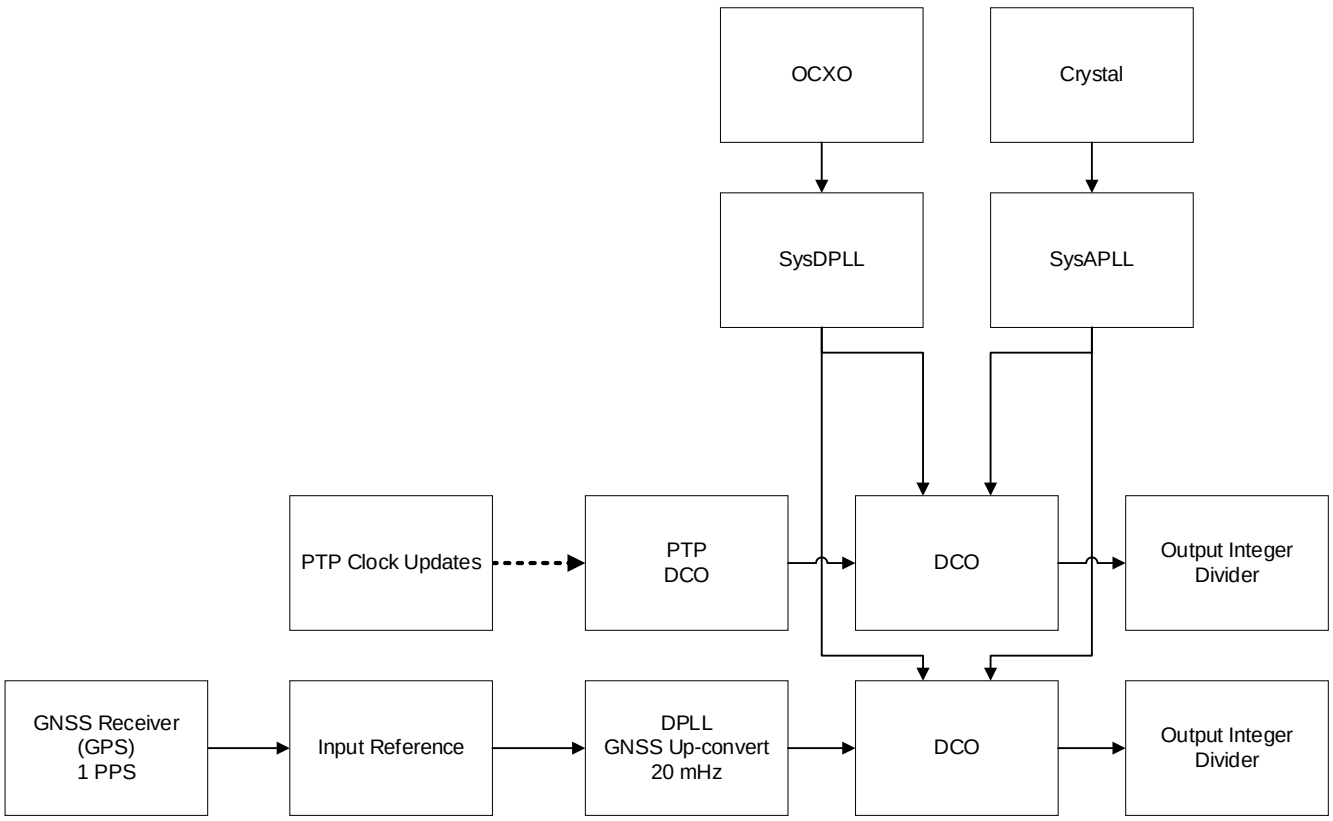

Figure 6. Example Bench Test Configuration

6. Revision History

Revision	Date	Description
0.1	May 10, 2021	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

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/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.