Renesas/Cyberon Speech Recognition

Cyberon Speech Recognition Technology

Target Device: **RA6, RA4, and RA2 Family of Devices**

Design Journey Purpose

The purpose of a design journey is to show, in a demonstrative way, how to incorporate the technology into a design. The base designs are by definition very simple to highlight the new technology and how it can be incorporated easily. Distinct lines are drawn between the application and the technology so the reader can more generalize how to incorporate it in their own design.

Introduction

Our goal with this Design Journey is to create a control for an LED-based light bulb based on the Renesas RA6/4/2 device family running Cyberon DSpotter for Voice User Interface (VoiceUI) technology. This technology is a non-connected speech recognition solution; it runs totally on the local MCU. While this limits its vocabulary and natural language support, it also allows for speech control of devices that are not connected to the cloud.

To accomplish this, we will design and build a circuit that can plug into any of the RA-Voice kit boards via the PMOD connector and write software to support the design. The software project accompanying this Design Journey is targeting the RA6 Voice Kit.

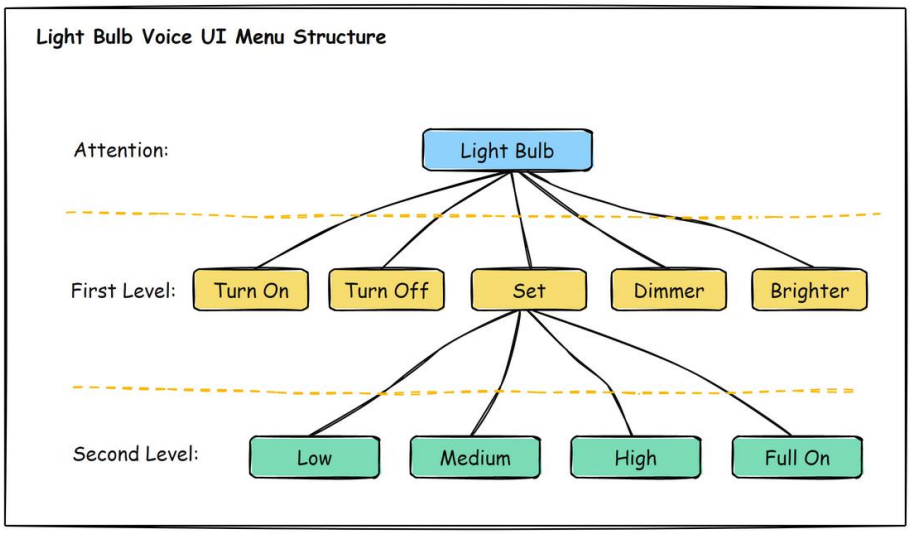
About This Design Journey

The purpose of this design journey is to show how the Renesas/Cyberon VoiceUI solution can be incorporated into a product design. Along with this document, the source code and project files, and the schematic PDF and gerber files of the plug-in board are available. Renesas also has available the schematic for the Voice kit boards for you to see how the microphones, comm, debugger, and GPIO are connected. At the time of publishing this design journey, these kits are available for sample on the Renesas website.

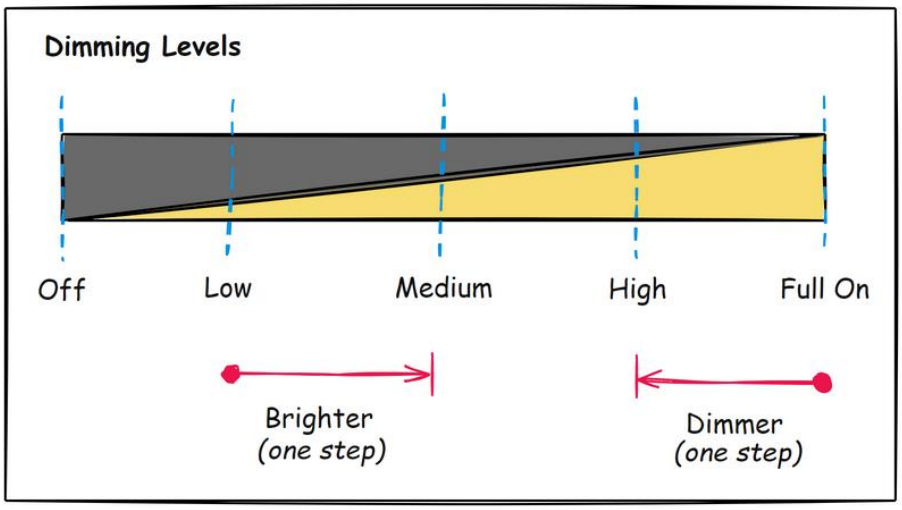
The Voice UI

One of the major differences between a connected, natural language solution, and a locally hosted speech recognition solution, is that a non-connected speech recognition system requires a stricter vocabulary set and syntax. This is because the interface is expecting certain words or phrases and cannot process natural language which can reorder words and still have the same meaning. This is the tradeoff for being able to run the speech recognition on an M-class device with a small memory footprint. . Multiple language support (Cyberon supports 44 at the time of publication of this document) can be achieved by loading multiple language sets and switching between them at runtime.

For our lightbulb, we have created a system that, depending on the desired outcome, will take two or three commands to complete a task. In graphic form, it looks like the following drawing:



As can be seen above, an attention word is first used to let the system know a command is coming. There are four commands (“Turn On,” “Turn Off,” “Brighter,” and “Dimmer”) that require no more information and complete the command. If the command word “Set” is used, then one of four preset levels is required next, to complete the command. “Dimmer” and “Brighter” traverse the presets one level at a time, and “on” and “off” move between extinguished and the last used preset level. Let’s look at what that means:



This defined command structure is easily created using the Cyberon DSpotter Modelling Tool and tested for recognition with their powerful command testing tools.

Unlike a “trained voice” solution, Cyberon has “pre-trained” the system for units of language sound, called phonemes. Each language has a set of phonemes. English, for example, has between 42 and 45 phonemes. Italian, by contrast, has 32-36. The DSpotter Modelling tool breaks down command text into these phonemes, which are pre-categorized. There is no need to take thousands of voice samples to train the system as you would with traditional speech recognition.

Creating the Lightbulb

I have two pet peeves about LED lighting: the first is that changing levels seems very unnatural – there is no “unit step function” in the real world yet changes in LED drive happen seemingly instantaneously. I have always asked my engineers to create a ramp when changing LED intensity to make it feel more “natural” – which probably means more like an incandescent bulb, who’s filament must change temperature as it changes brightness. Which leads me to my second pet peeve: color temperature. As incandescent lights are dimmed, the filament changes color, creating a warmer light as it dims, and cooler light as it gets brighter. We can recreate this effect by color mixing warm and cool LEDs. “Mixing” means driving the two LEDs at different average currents, by having individual PWMs for each LED.

Starting Specification

So, at this point we have created a rough specification:

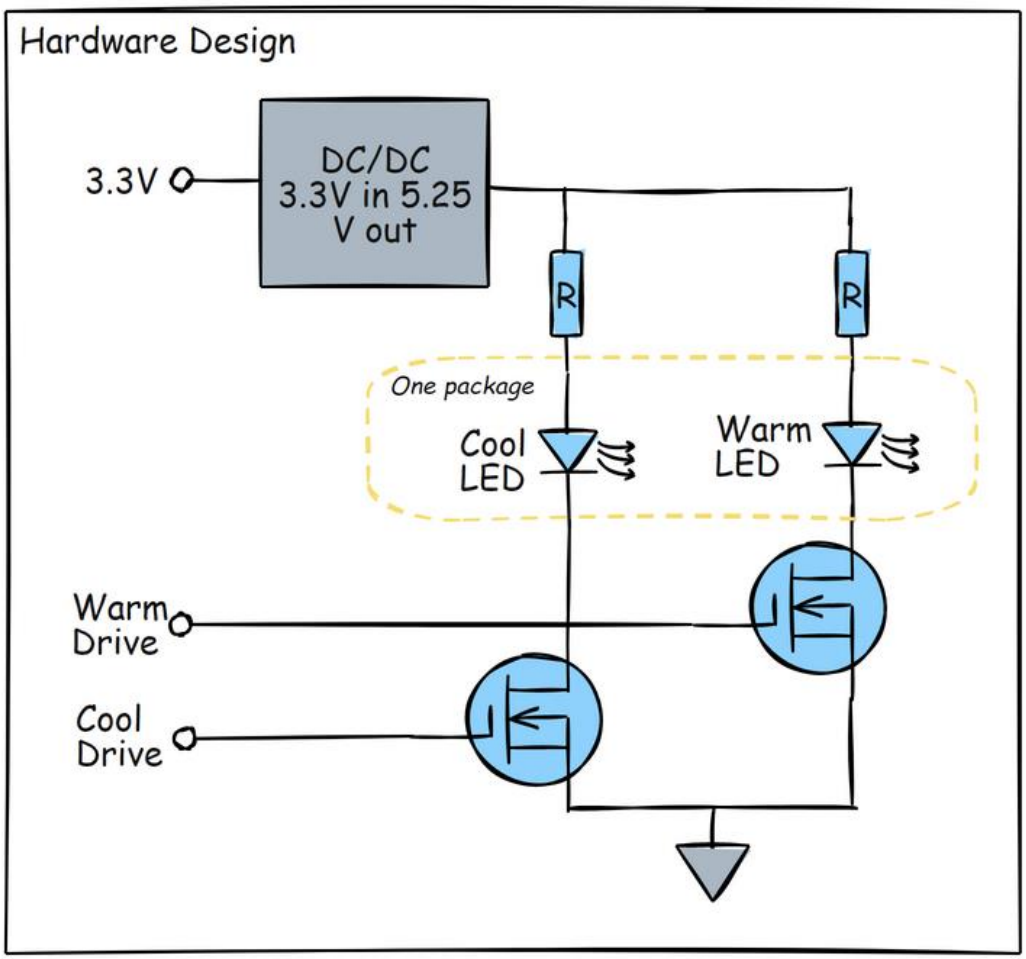
- LED lights, dimmed by PWM
- Warmer color when dim, cooler color when bright
- Four preset intensities
- Changes in intensity are ramped
- “Off” dims from the current level to off, “On” returns to the previous on level
- “Brighter” and “Dimmer” move between intensities one at a time

Note that these are standard specs that apply to any bulb you may find in the market.

Hardware Design

The hardware design for the bulb is relatively simple because the RA devices have timers with PWM outputs which can drive transistor gates directly. The use of transistors is driven by two things: firstly, the GPIO pins of any processor cannot handle the currents required to drive an LED at illumination levels, secondly, the forward voltage drop of white LEDs is very close to the 3.3V Vcc used by the processor. Typically, the LEDs are driven with a higher voltage. The processor pins cannot tolerate this higher voltage, so to isolate the I/O pins from both this voltage and current requirement, we can use small MOSFETs to drive the LEDs.

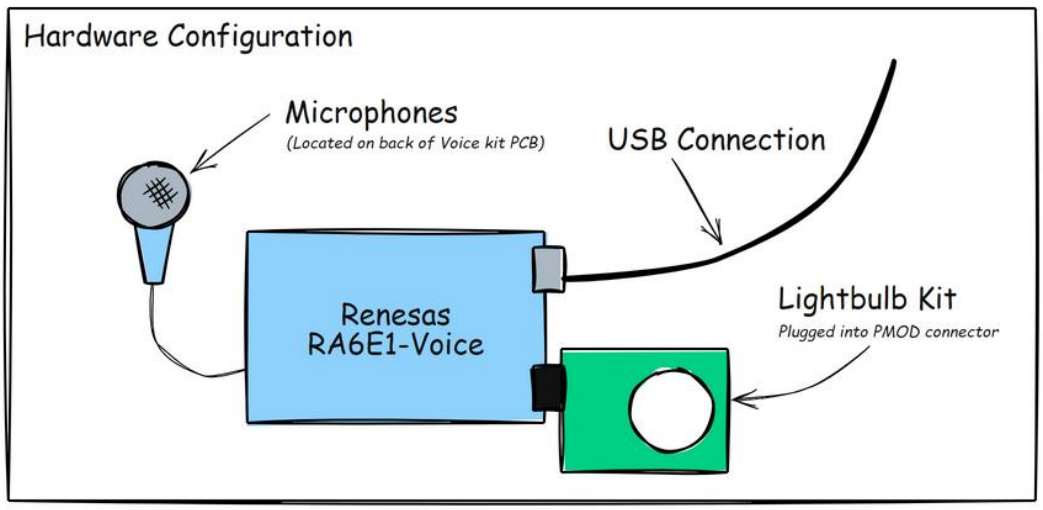
Since the PMOD connector has ground and 3.3V, we must create a higher voltage. For this, we are using a Renesas part on our plug-in board, the ISL9111A buck-boost controller, with the output set to 5.25VDC. The following illustration shows the overall design of the LED plug-in board.



We will drive the two transistor gates with the output from a PWM running with a carrier frequency of 4kHz. This is well above the optical detection of the eye – even in the periphery where we are more sensitive, and well below where the switching transitions occupy a high percentage of overall time (this, in the end, is what makes transistors run hot – not what percentage of time they are on, but the percentage of time they spend in the linear region during switching).

The hardware for the processor and its peripherals is the one supplied with the voice kit. It has a processor, two digital and two analog mics, and a PMOD connector. Our LED board plugs into the PMOD connector. The other connection is the USB-micro OB to the PC for debug and serial monitoring (the JLINK-OB debugger creates a VCP along with the debug connection).

For a schematic of the voice kit PCB, please refer to the documentation page for that kit. For a schematic of the lightbulb plug-in PCB, see the accompanying documentation for this Design Journey.



Dimming

Dimming is a funny thing: done well it looks smooth and natural, done poorly and it feels very uncomfortable. “Poorly” means uneven steps in either brightness or timing. At first, I wondered if I’d need to use an RTOS to be able to run the Cyberon data collection algorithm and my dimming at the same time.

I looked at doing the dimming in the interrupt – it really isn’t too bad if I maintain a small table of dimming values on a continuum. With the table I’d be able to just count, index, and output. That’s a pretty small amount of work to be done (I try to minimize the amount of work that gets done in interrupts).

But then we realized that with the Renesas peripherals and connection fabric on the device, I could possibly do it all in hardware, so we looked into it. (See sidebar: “Making Hardware Work for You.”)

Using the built-in hardware on the RA device, I was able to create a hardware chain with two timers, memory, DMA, and the event link controller (ELC). By adjusting the start/stop points in one long table of 400 entries, I was able to create smooth dimming in the background by just setting the targets in the command processing callback of the Cyberon code.

The following diagram shows how the selected peripherals work together to create a hardware-based, smooth dimming “algorithm” that is not dependent on firmware timing to run. A simple set of register sets and bit sets allows the hardware to take over and smoothly transition from one level to the next. The DMA is used to update the duty cycle of the PWM at regular intervals until the next dimming target value is met. In our design, we change the PWM value every 10 PWM cycles, with 100 steps between presets. That means it takes 1,000 PWM cycles to change from one setting to the next – less than a second.

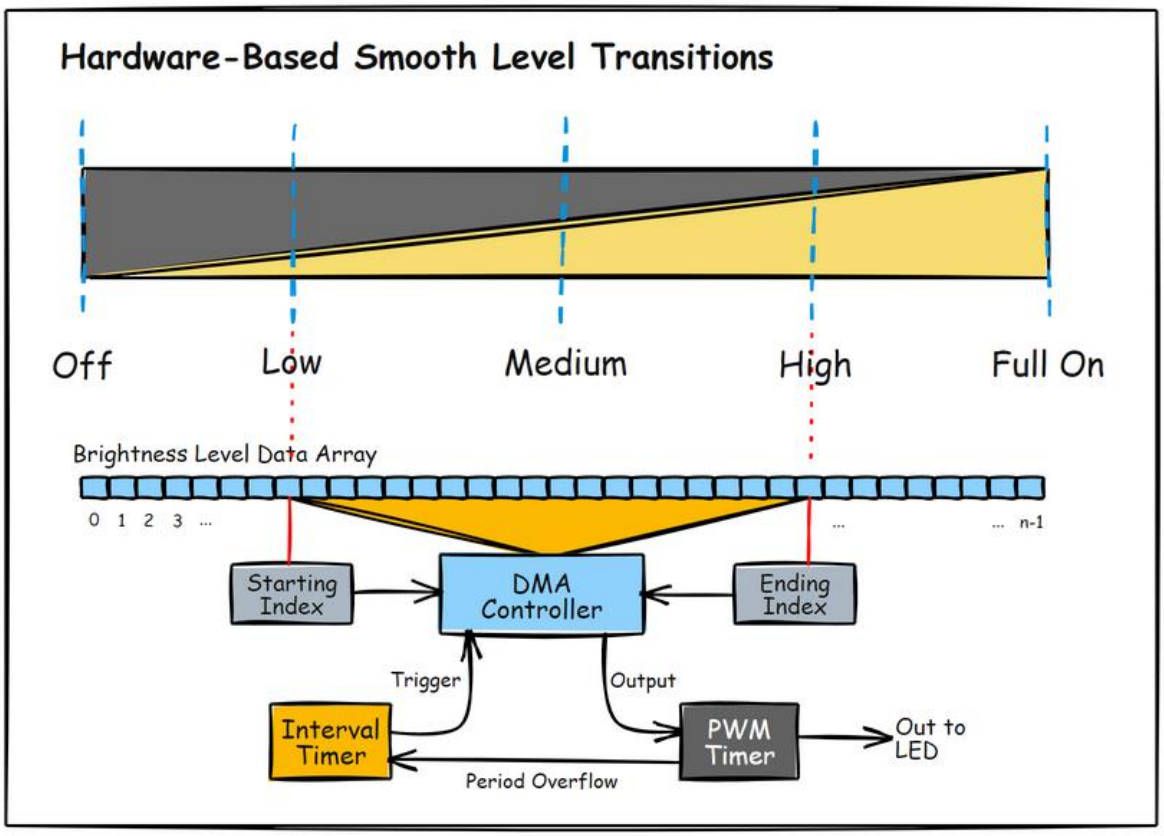
MAKING HARDWARE WORK FOR YOU

One of the great things about MCUs is how peripherals can share high-speed interfaces and connect with each other to accomplish tasks with minimal, if any, firmware intervention, simply because they are integrated into the architecture.

Renesas has always had a rich set of high-performance peripherals: both digital and mixed mode. Together with the Event Link Controller (ELC) in the RA family, these peripherals can be configured to do many functions independently.

In this design, the use of a linked hardware solution makes it easy to do two time-dependent tasks together without the overhead of an RTOS to manage the processes.

While this solution generates data for the outside world (the PWMs) the same opportunity is there for data acquisition with digital and analog peripherals.

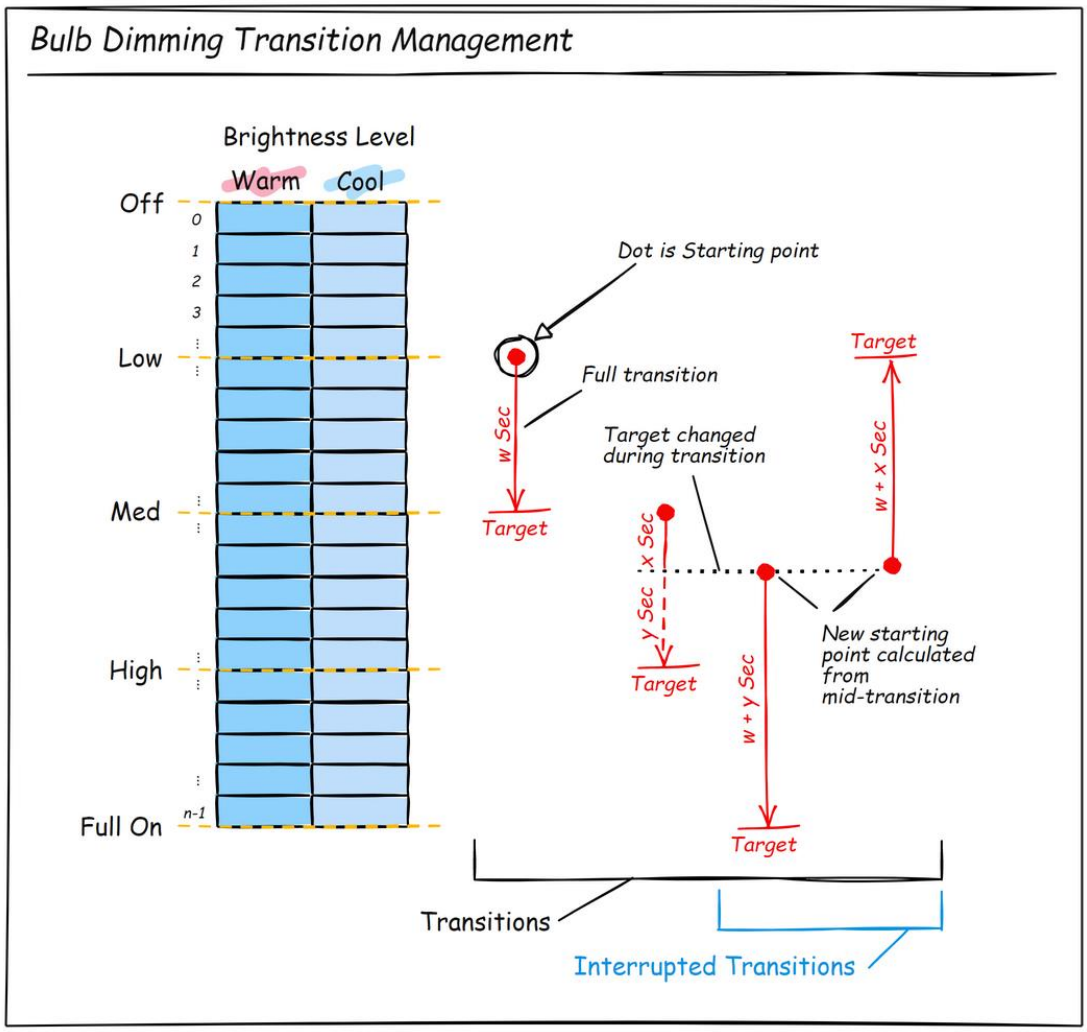


The diagram shows how the interval timer, clocked by the period overflow of the PWM timer, triggers a DMA transfer into the PWM duty cycle threshold register within the output timer. The firmware simply sets the start and stop index which correspond to the current dim level and the target dim level, respectively, and enables the transfer. The hardware then takes over and traverses the table. The transition time between presets is fixed. (You could increase “sophistication” of the system by changing the number of period overflows required to launch a transfer if the bulb is moving more than one preset, say low to high. We did not do that in our design.)

So, in the end, we had three possible paths: RTOS with code implementation, Interrupt code, or hardware implementation. The clear winner in this case was the hardware implementation of the dimming. It allows the dimming to take place with a few lines of code to set up, and then no CPU cycles.

The Data Table

Since we have four dimming levels: low, medium, high, full-on, and we want to move between them, a table of 400 entries gives us 100 dimming steps between. By making it 401 entries we can also have a zero-output starting point. This drawing below shows how that would work.

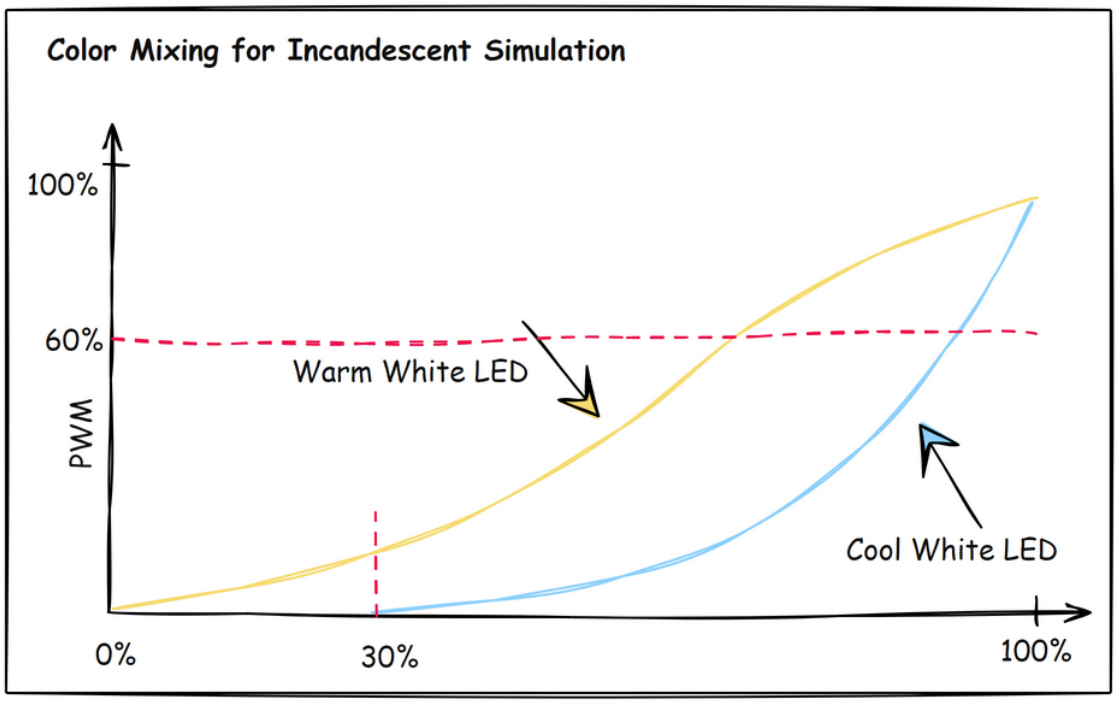


As you can see, because of my need to do smooth dimming, there is the possibility that a command can interrupt a dimming in progress, so in that case the dimming "path" will be more than the 100 steps between settings. But it will eventually settle on the correct value.

Now let's talk about the data table contents. The eye is not linearly responsive to light, so instead of a linear curve, we can use a 3rd order curve, reducing the changes at low light and increasing them at the bright end. Since we want the light to be warmer at the beginning, we can simply delay by 30% and accelerate the cool LED. If we were dimming under FW control, we could create two one dimensional arrays, a two-dimensional table with one dimension being the dimming level and the other being warm and cool values for each level, or a one-dimensional array of structures, each structure having a warm and cool value. For DMA, the two-table approach is more straightforward, so there is a source array for each of the DMA transfers.

[Note: We generated the table in SRAM during runtime for this Design Journey. It would normally be predetermined and located as a constant in flash. We did it this way so the user could easily change the tables and try different curves. Moving the table to flash does not change anything about the implementation or the performance.]

The curve of the table data would look like this:



1.1 Setting up the Project

If you've watched the lab video for the Renesas with Cyberon solution (and why wouldn't you !) you've seen how easy it is to set everything up. But I will outline the process here. It is a combination of hardware configuration using the renesas FSP Configurator, and the inclusion of some Cyberon Libraries and Data files. The process is the following (most of this is covered in the VoiceUI lab video located on the landing page of the Cyberon partner solution):

1) Ensure you have the correct CMSIS pack for the FSP version you will be working on (this Design Journey runs on 3.8)
2) Start a new project, picking the latest FSP and choosing the RA6E1_Voice BSP, no RTOS, Bare Metal, and flat (no trust zone).
3) Using the configurator, set up the following stacks and associated peripheral properties:
    a. UART
    b. GPIO
    c. Flash
    d. SPI and two timers working together to simulate I2S reception (this, again, is the hardware working *for* you to create something that does not exist on the part)
    e. Stack and Heap sizes
4) Generate code

[This process is detailed in the lab notes available on the Renesas GitHub site, and in the lab video.]

The above four steps set the system up for using the Cyberon Solution. We then need to set up the peripherals for dimming. Note that the BSP for these boards set the GPIO as output pins, not PWM pins, because the demo code just turns LEDs on and off.

To Set up the dimming via hardware, using the "Stacks" tab, add two stacks for general purpose timers:
[New Stack | Timers | Timer, General PWM (r_gpt)]
and two for DMA:
[New Stack | Transfer | Transfer (r_dmac)].

The two timer blocks are used for PWM (two channels of one timer are used for output) and the cadence counter for dimming (every 10 PWMs). The two DMA channels are used to transfer from memory (the dimming table) to the registers for PWM duty cycle. There are two DMA channels even though they run at the same time because there are two target PWM registers to be updated.

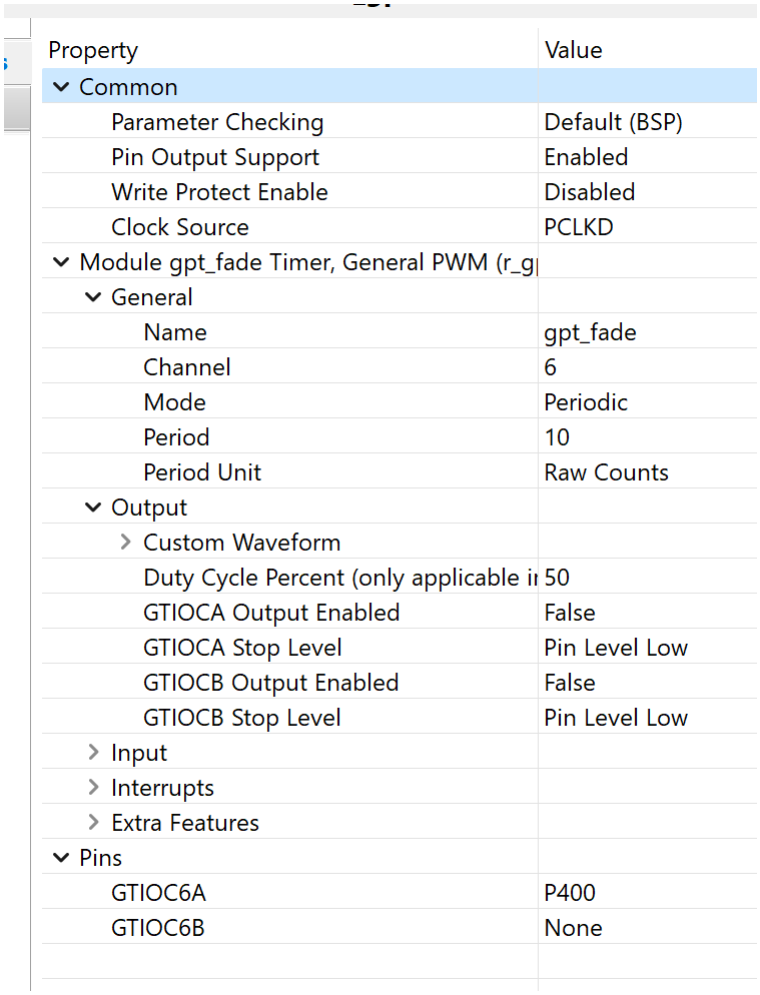
Name these timers "gpt_led" and "gpt_fade." Name the two DMA channels "dma_warm" and "dma_cool'" and then set the properties as follows:

gpt_led:

Property	Value
Common	
Parameter Checking	Default (BSP)
Pin Output Support	Enabled
Write Protect Enable	Disabled
Clock Source	PCLKD
Module gpt_led Timer, General PWM (r_gpt	
General	
Name	gpt_led
Channel	7
Mode	PWM
Period	4000
Period Unit	Hertz
Output	
Custom Waveform	
Duty Cycle Percent (only applicable i	0.01
GTIOCA Output Enabled	True
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	True
GTIOCB Stop Level	Pin Level Low
Input	
Interrupts	
Extra Features	
Pins	
GTIOC7A	P304
GTIOC7B	P303

[We are using timer channel 7 because the two outputs are connected to the GPIO pins that go to the PMOD connector – one for the cool and one for the warm LED.]

gpt_dim:



Property	Value
Common	
Parameter Checking	Default (BSP)
Pin Output Support	Enabled
Write Protect Enable	Disabled
Clock Source	PCLKD
Module gpt_fade Timer, General PWM (r_g	
General	
Name	gpt_fade
Channel	6
Mode	Periodic
Period	10
Period Unit	Raw Counts
Output	
Custom Waveform	
Duty Cycle Percent (only applicable i	50
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Input	
Interrupts	
Extra Features	
Pins	
GTIOC6A	P400
GTIOC6B	None

dma_warm:

Property	Value
Common	
Parameter Checking	Default (BSP)
Module dma_warm Transfer (r_dmac) GPT6	
Name	dma_warm
Channel	1
Mode	Normal
Transfer Size	4 Bytes
Destination Address Mode	Fixed
Source Address Mode	Offset addition
Repeat Area (Unused in Normal Mode)	Source
Destination Pointer	&R_GPT7->GTCCR[3]
Source Pointer	NULL
Number of Transfers	1
Number of Blocks (Valid only in Repeat,I	0
Activation Source	GPT6 COUNTER OVERFLOW (Overflow)
Callback	NULL
Context	NULL
Transfer End Interrupt Priority	Disabled
Interrupt Frequency	Interrupt after all transfers have completed
Offset value (Valid only when address m	4
Source Buffer Size	1

dma_cool:

Property	Value
Common	
Parameter Checking	Default (BSP)
Module dma_cool Transfer (r_dmac) GPT6 C	
Name	dma_cool
Channel	0
Mode	Normal
Transfer Size	4 Bytes
Destination Address Mode	Fixed
Source Address Mode	Offset addition
Repeat Area (Unused in Normal Mode)	Source
Destination Pointer	&R_GPT7->GTCCR[2]
Source Pointer	NULL
Number of Transfers	1
Number of Blocks (Valid only in Repeat,I	0
Activation Source	GPT6 COUNTER OVERFLOW (Overflow)
Callback	NULL
Context	NULL
Transfer End Interrupt Priority	Disabled
Interrupt Frequency	Interrupt after all transfers have completed
Offset value (Valid only when address m	4
Source Buffer Size	1

The code to manage the DMA (start/stop, set start and end addresses, etc.) is pretty straightforward, with the "twist" of accounting for interrupted fades:

```
void led_set_fade(const uint16_t target)
{
    /* Stop the timer triggering duty cycle updates, if any are still in progress */
    R_GPT_Stop(&gpt_fade_ctrl);

    /* Correct current stage, based on old target and transfers remaining */
    transfer_properties_t dma_info = {0};
    R_DMAC_InfoGet(&dma_cool_ctrl, &dma_info);

    if (led_stage < led_target) // stage WAS going up
    {
        led_stage = led_target - (uint16_t) dma_info.transfer_length_remaining;
    }
    else // stage WAS going down
    {
        led_stage = led_target + (uint16_t) dma_info.transfer_length_remaining;
    }

    /* Don't start DMA if we're already at target duty cycle */
    if (target == led_stage)
    {
        return;
    }

    led_target = target;

    transfer_info_t * p_cool = dma_cool_cfg.p_info;
    transfer_info_t * p_warm = dma_warm_cfg.p_info;

    if (led_stage < target) // stage WILL BE going up
    {
        /* Calculate and set the number of steps to the new target */
        p_cool->length = target - led_stage;
        p_warm->length = target - led_stage;

        /* Set the transfer addressing mode */
        p_cool->src_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED;
        p_warm->src_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED;

        /* Set the first address to start the DMA with */
        p_cool->p_src = led_ramp_cool + led_stage + 1;
        p_warm->p_src = led_ramp_warm + led_stage + 1;
    }
    else // stage WILL BE going down
    {
        /* Calculate and set the number of steps to the new target */
        p_cool->length = led_stage - target;
        p_warm->length = led_stage - target;

        /* Set the transfer addressing mode */
        p_cool->src_addr_mode = TRANSFER_ADDR_MODE_DECREMENTED;
        p_warm->src_addr_mode = TRANSFER_ADDR_MODE_DECREMENTED;

        /* Set the first address to start the DMA with */
        p_cool->p_src = led_ramp_cool + led_stage - 1;
        p_warm->p_src = led_ramp_warm + led_stage - 1;
    }

    /* Reconfigure DMAs with new settings and start the timer to begin transfers */
    R_DMAC_Reconfigure(&dma_cool_ctrl, p_cool);
    R_DMAC_Reconfigure(&dma_warm_ctrl, p_warm);

    R_GPT_Start(&gpt_fade_ctrl);
}
```

To see how to use it, and how to set up the timers to be running, as well as some code to generate the LED curves at runtime, please take a look at the source code (specifically, the hal_entry.c file).

The current state of the light (which level it is at or was before an off command) a modular variable is used.

You can see that in the voice command callback, the fading control function is called at the appropriate endpoints. To make the code easier to read and maintain, some modular constant tables are also declared. This makes it very easy to change the values of mid, low, hi all in one place.

Conclusion

As you can see from this design journey, incorporating the Renesas/Cyberon VoiceUI into your own product is fast, easy, straightforward, and very maintainable.

Voice Kit used throughout this journey can be requested from the links below.

Introductory and Lab videos also ready for you.

TW001-VUIA6E1POCZ - RA6E1 Voice User Reference Kit | Renesas

TW001-VUIA4E1POCZ - RA4E1 Voice User Reference Kit | Renesas

TW001-VUIA2L1POCZ - RA2L1 Voice User Reference Kit | Renesas

Revision History <revision history>

Rev.	Date	Description	
		Page	Summary
1	14-Jul-23		First edition

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

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/.