Chapter 5: Cyberon

After completing this chapter, you will understand how to use the Cyberon machine learning solution to create pre-trained and optimized one and two-stage audio keyword detection models for use with Infineon MCUs.

Table of contents

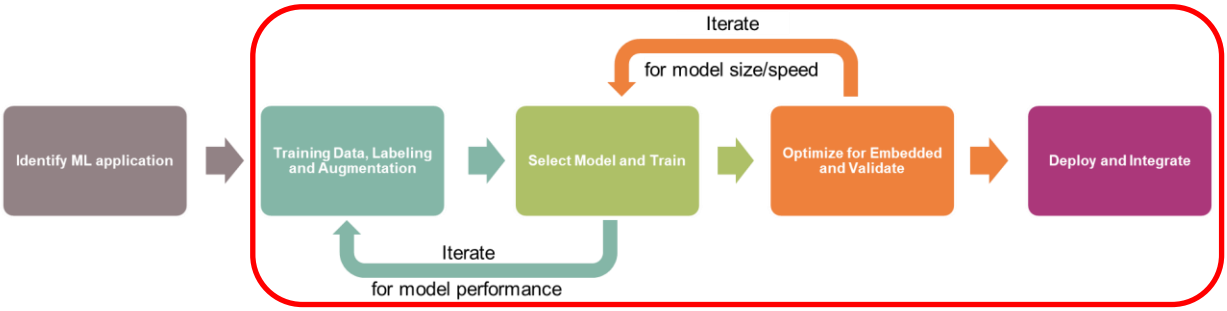
5.1 Overview 2
5.2 Getting Started 3
5.2.1 Create and run a demo application 3
5.2.2 Obtain license file 5
5.2.3 Run demo with license installed 6
5.2.4 Create a custom model 6
5.2.5 Test the model online 9
5.2.6 Import custom model and run in demo application 11
5.2.7 Creating a two-stage custom model 11
5.3 Firmware API 12
5.3.1 Cyberon library files 12
5.3.2 User code 12
5.4 Exercises 14
Exercise 1: Create a custom one-stage keyword detection application 14
Exercise 2: Create a custom two-stage keyword detection application 15

Document conventions

Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

5.1 Overview

In this chapter, we will be using one of Infineon's Machine Learning partners, Cyberon to produce a fully-trained machine learning model that can recognize one or two-stage keywords using their DSpotter Trigger-word model. It supports more than 30 languages and even multiple dialects of some languages such as American and British English!



One-stage models recognize a trigger word (known as a "wake word") while two-stage models recognize a trigger word followed by additional command words. For example:

Model Type	Trigger Word	Command
One-stage	Hello Cyber Voice	N/A
Two-stage	Hello Cyber Voice	Play music Pause music Next track Volume up Volume down

On a two-stage model, once the trigger word is recognized, you can use one or more commands until a timeout occurs. The default timeout in the demo applications occurs 10 seconds after the last recognized keyword.

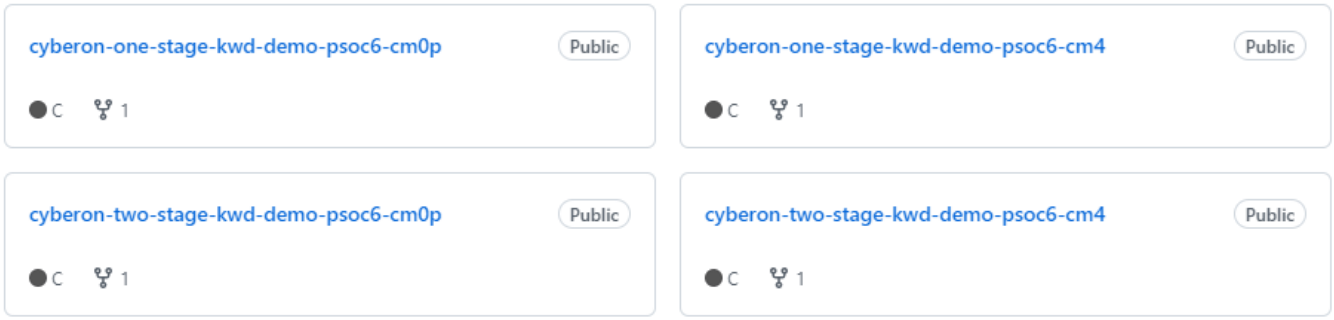
Once the model has been created, it can be integrated into a PSoC™ 6 application. ModusToolbox™ code examples are provided to run the keyword detection model on either the CM0+ or CM4 CPU in a PSoC™ 6 device (see https://github.com/CyberonEBU). As you will notice, the Cyberon API lets you register a callback function that is called whenever the keyword(s) are detected, allowing for simple integration into an end product application.

5.2 Getting Started

Since the Cyberon flow is fairly straight-forward, we'll jump right into discussing how to use it on a PSoC™ 6.

5.2.1 Create and run a demo application

The simplest way to use the Cyberon solution is to clone or download one of their GitHub code example repos (https://github.com/CyberonEBU). Then, create an application in ModusToolbox™ using it as the application template by browsing to its location on the application selection page. There are four code example repos to choose from:

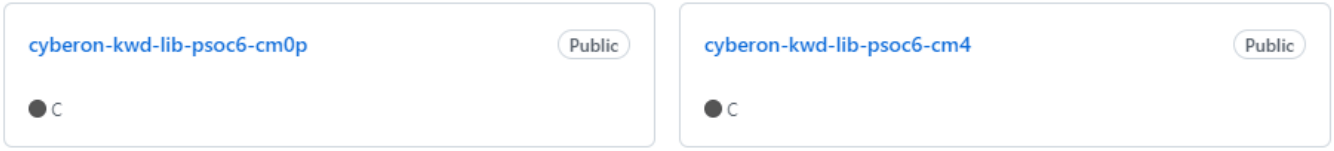


As you can tell from the repo names, you can choose whether you want to run the model on the CM0+ or the CM4 CPU and whether you want one-stage or two-stage keyword detection.

The CM0+ code examples are dual-core applications that run the keyword detection engine on the CM0+, leaving the CM4 free for user code.

The CM4 code examples are single-core applications that run the keyword detection engine on the CM4.

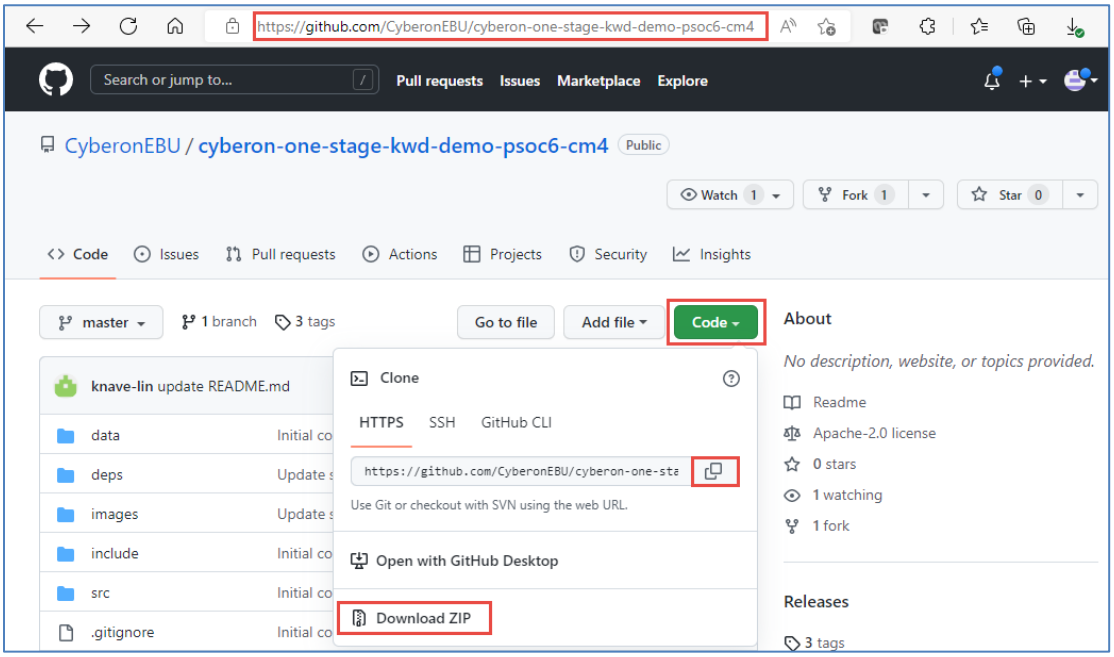
Note: *There are also two library repos – one of these libraries will be included automatically by each code example depending on which CPU you choose to run the model on.*



As an example, here are the detailed steps to download, create and run the one-stage keyword detection demo for the CM4:

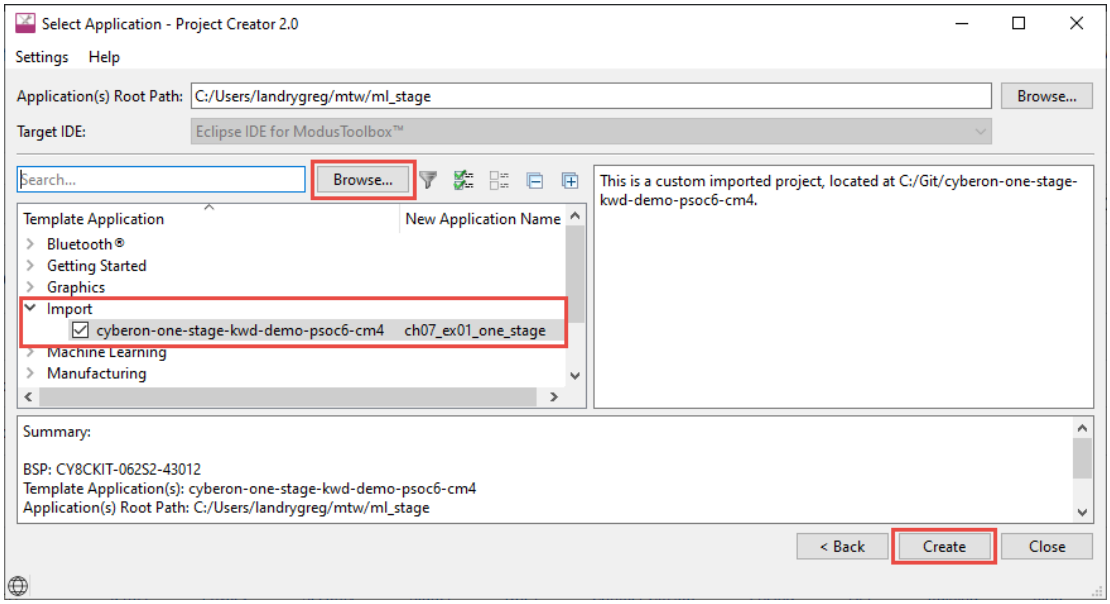
5.2.1.1 Get the demo

The first step is to clone or download the repo. From the GitHub page, click the green button that says **Code** and either copy the URL to use in a git clone operation or click the Download ZIP link. If you download the ZIP file, unzip it once the download has completed.



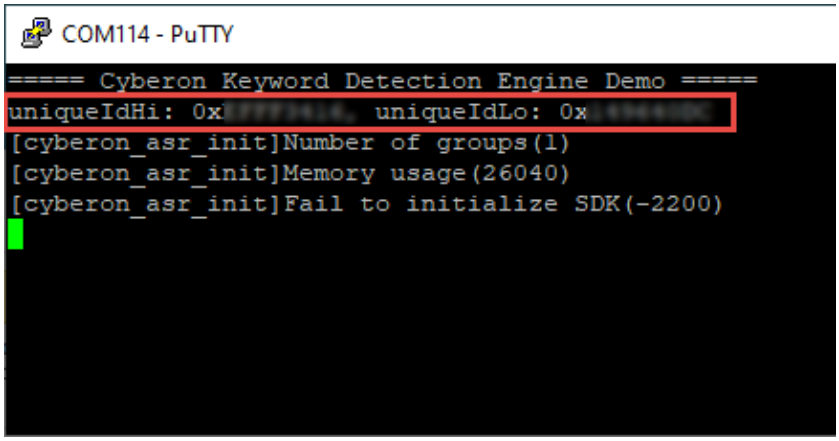
5.2.1.2 Create the application

Once the repo is available, run Project Creator and select your kit on the first page as usual. On the second page, click the Browse… button and navigate to the location of the repo on disk. Select that as the template application and change the name if desired.



5.2.1.3 Program the kit

Plug in your kit and open a UART terminal window. Build it and program the ModusToolbox™ application that you created to the device. The application will not work yet because you don't have a license for the Cyberon DSpotter Trigger-word model. You will need the `uniqueIdHi` and `uniqueIdLo` values listed in the UART terminal for your kit to get a free evaluation license.

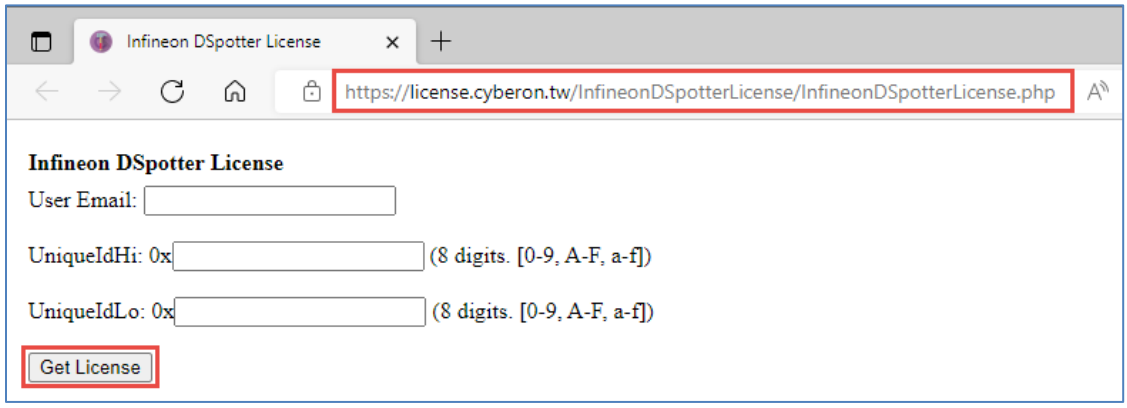


5.2.2 Obtain license file

Now that you have the IDs for your kit, you can get a free evaluation license for that kit. A link is provided in the application's README.md file, but the direct link is:

https://license.cyberon.tw/InfineonDSpotterLicense/InfineonDSpotterLicense.php

From that site, you should enter your email address and the two IDs that you received from the UART terminal during the previous step.



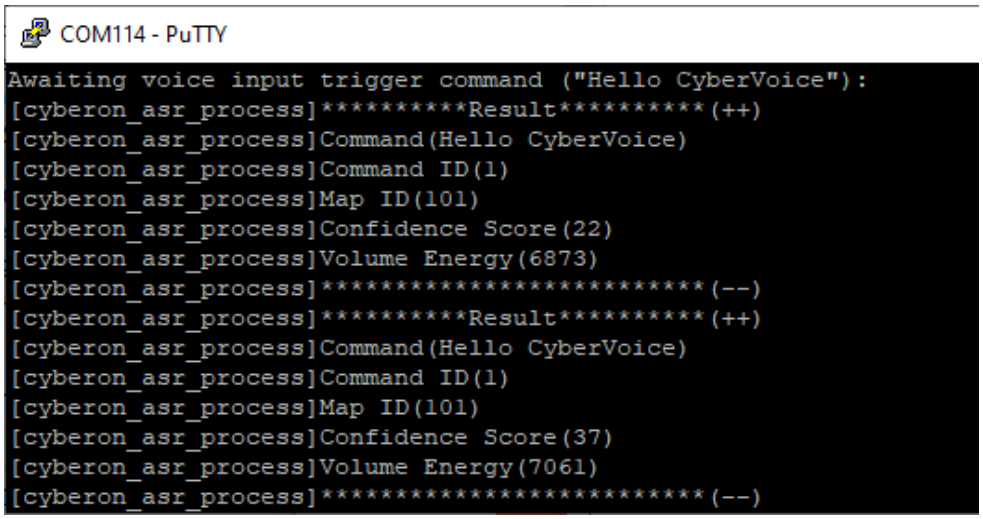
Note: *The IDs are unique to a particular PSoC device so you can use the same license file for any Cyberon DSpotter application on that kit, but if you want to use an application on a different kit in the future, you will need to get a license file for that kit.*

Once you complete the form and click Get License, a license file will be emailed to you. Just save the file and replace the file *data/CybLicense.bin* inside the application.

Note: *The name must be CybLicense.bin. You will need to rename the file that was emailed to you.*

5.2.3 Run demo with license installed

Once you have replaced the *CybLicense.bin* file with your own, you must clean the application and then rebuild it. If you don't do a clean operation first, the new license file will not be picked up. Program the application and look at the UART to see the results. Say the trigger word (and other words) a few times to see how well it distinguishes the correct word.



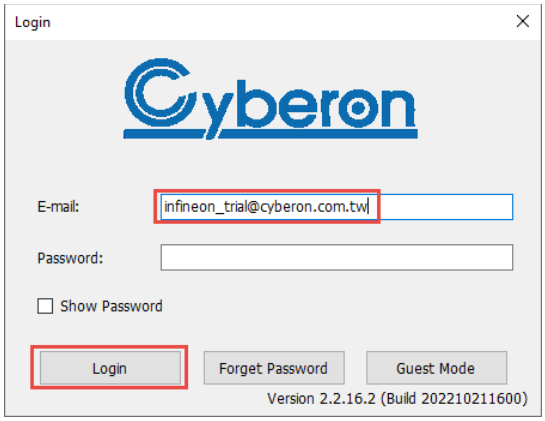
5.2.4 Create a custom model

Next, let's create a custom model with our own trigger word. The README.md file in the application has a link to the Cyberon DSMT (DSpotter Modelling Tool), but here is the direct link for reference:

https://tool.cyberon.com.tw/DSMT_V2/index.php?lang=en

You should have already installed the tool in chapter 1, but the documentation on the page may be useful for reference.

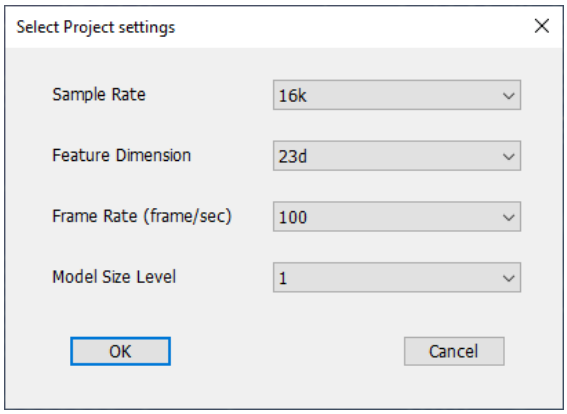
When you launch the tool, you will get this initial window. Use infineon_trial@cyberon.com.tw as the email address, leave the password blank and click the **Login** button.



Note: DSMT is a Windows only program.

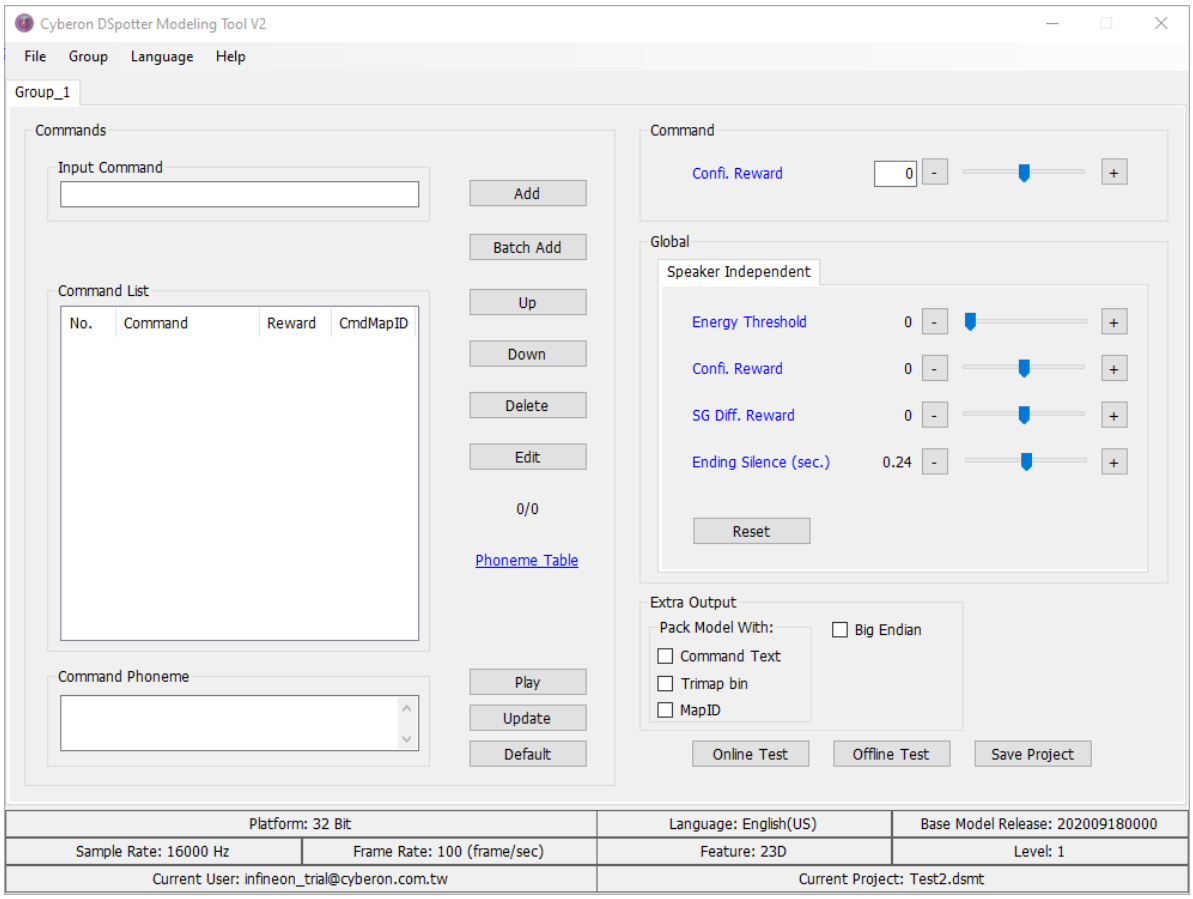
Note: DSMT requires an internet connection to create models.

In the tool, create a new project (**File > New Project…**). Give it a name and select the language that you want to use for the model. Observe that there are English versions optimized for different countries. On the next page, leave the default project settings as-is:



Select a location to save the project on disk. It will create a sub-directory with your project name in the location that you specify.

When the project opens, you will see a window like this:



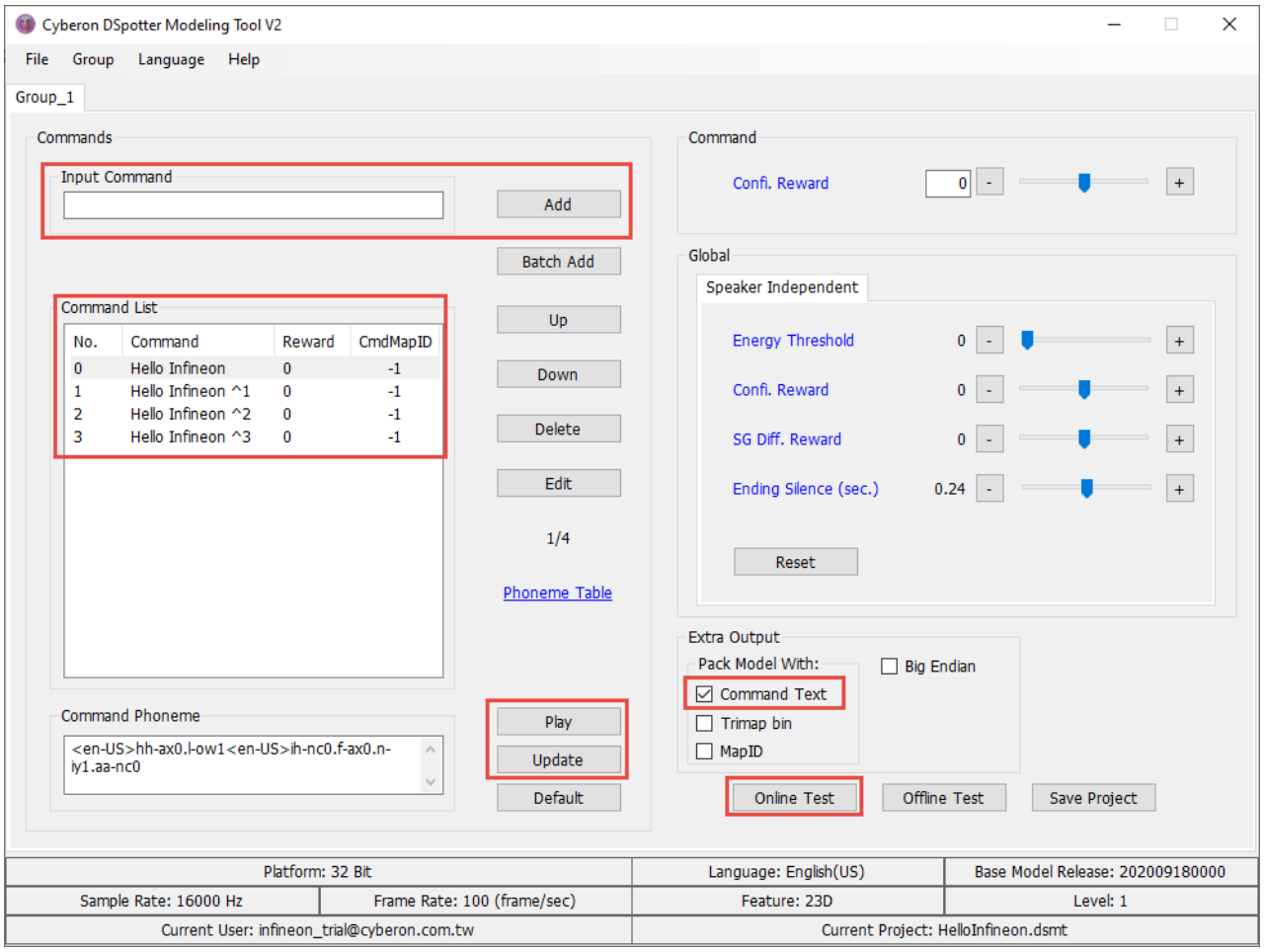
From this window, type in your trigger word or phrase in the Input Command box and click **Add**. This will add one or more commands to the command list.

You might see more than one command because the tool has several alternate pronunciations for certain words. For example, the word "Hello" is sometimes pronounced more like "Hallo." You can hear the different pronunciations by selecting one of the commands and clicking the "Play" button.

You can change the pronunciation of any of the commands by modifying the Command Phoneme box and clicking **Update**.

Another way to add additional pronunciations is to type them into the Input Command list phonetically. For example, I used both "Hello Infineon" and "Hello Infiniyen" as spellings to get the four entries you see below.

Note: *I didn't want the command name to show up as "Hello Infiniyen" so after I added it, I added a second instance of "Hello Infineon" and then copied the command phonemes from "Hello Infiniyen" to the new commands and updated them. Once that was done, I deleted the "Hello Infiniyen" versions of the command.*



Note that the value of CmdMapID is the same for all commands in this window. That means that any of these commands will be reported in the firmware as the same command. This is how multiple pronunciations get mapped to the same command. When you do a two-stage model, the command CmdMapID value for each independent command will be a different value.

On the right side of the window, there is a Confidence Reward setting for each command and global settings for Energy Threshold, Confidence Reward, SG Difference Reward, and Ending Silence. These are used to adjust

how sensitive the model is to the words that it is detecting and can be used to trade-off between false negative and false positive results. The definitions are:

- **Confidence Reward (command)**： A larger value means the command can be more easily recognized. On the other hand, it could also increase the risk of false positives. Recommended range: -10 to 10.
- **Energy Threshold** ： Sets the minimal energy for a recognized command to reject falsely recognized commands caused by background noise of relatively low volume. Recommended range: 0 to 200.
- **Confidence Reward (group)**： Decreasing this value will inhibit the false alarms, but will make the model less responsive to users' commands. The engine will add group and command confidence rewards to get the confidence score offset. Recommended range: -10 to 10.
- **SG Difference Reward**： An indication of how much the voice is different from Silence/Garbage. Decreasing the SG Difference Reward will inhibit the false alarms, but will make the model less responsive to users' commands. Recommended range: -10 to 10.
- **Ending Silence**： Defines the duration of silence in seconds after voice infput for the engine to determine the end of a voice command.

Before saving the model, you should ensure that the **Extra Output > Pack Model With > Command Text** box is checked. This provides text that the firmware can use to print the command names. Otherwise, the firmware will only know the IDs.

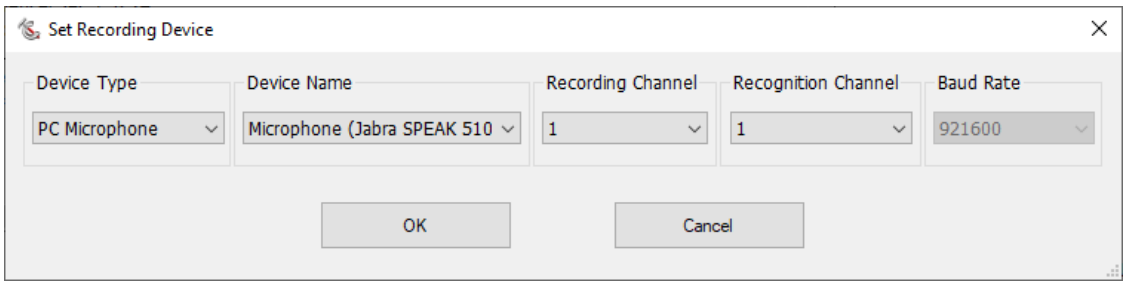
Once you are done making changes, click **Save Project** to save your settings and generate all of the files you will need to use the model on a PSoC™ 6.

5.2.5 Test the model online

Once the model is set up, click **the Online Test** button to launch the test utility.

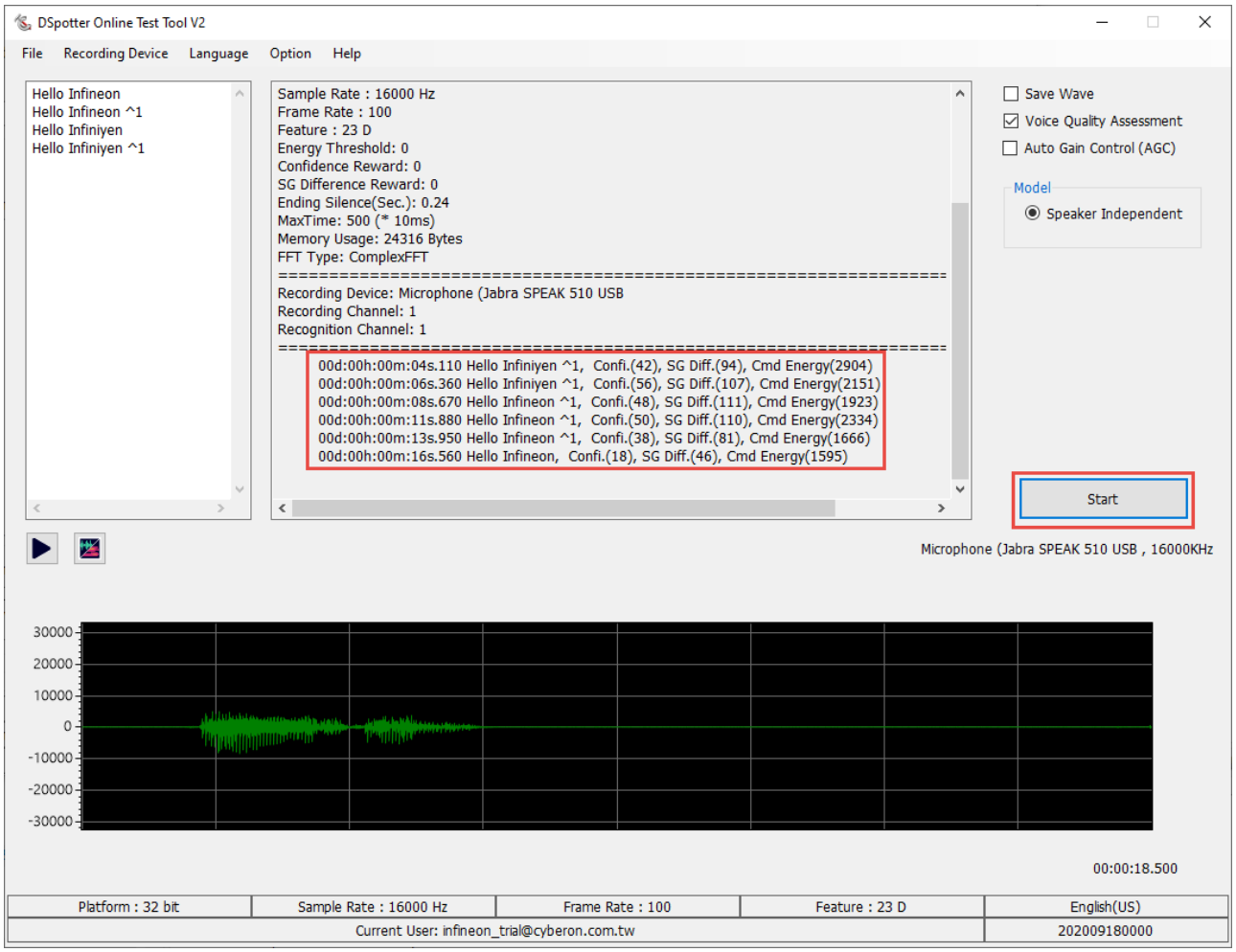
Note: *You must install the online tester for this to work.*

Once the tool opens, click **Recording Device** in the menu to select the microphone that you want to use.



Once the recording device is connected, click the **Start** button to start recording. When the model recognizes the trigger word, you will see it displayed along with the Confidence, SG Diff and Energy values. It also shows which of the different pronunciations matched. Click **Stop** when you are done.

You can play back what you recorded by pressing the triangular play button on the left side of the window if you want to hear what was said for each recognized word again. You can also save the recording by using **File > Save Wave…** from the menu. This allows you to use the same recording on different models to compare their results. The saved waves can be used by the Offline test tool.



Based on results from testing, you can adjust the parameters in the SDMT to make it perform to your expectations. When you are happy with the model, click **Save Project** again.

5.2.6 Import custom model and run in demo application

Once the model is complete, importing it into the demo application is very straightforward. The process is:

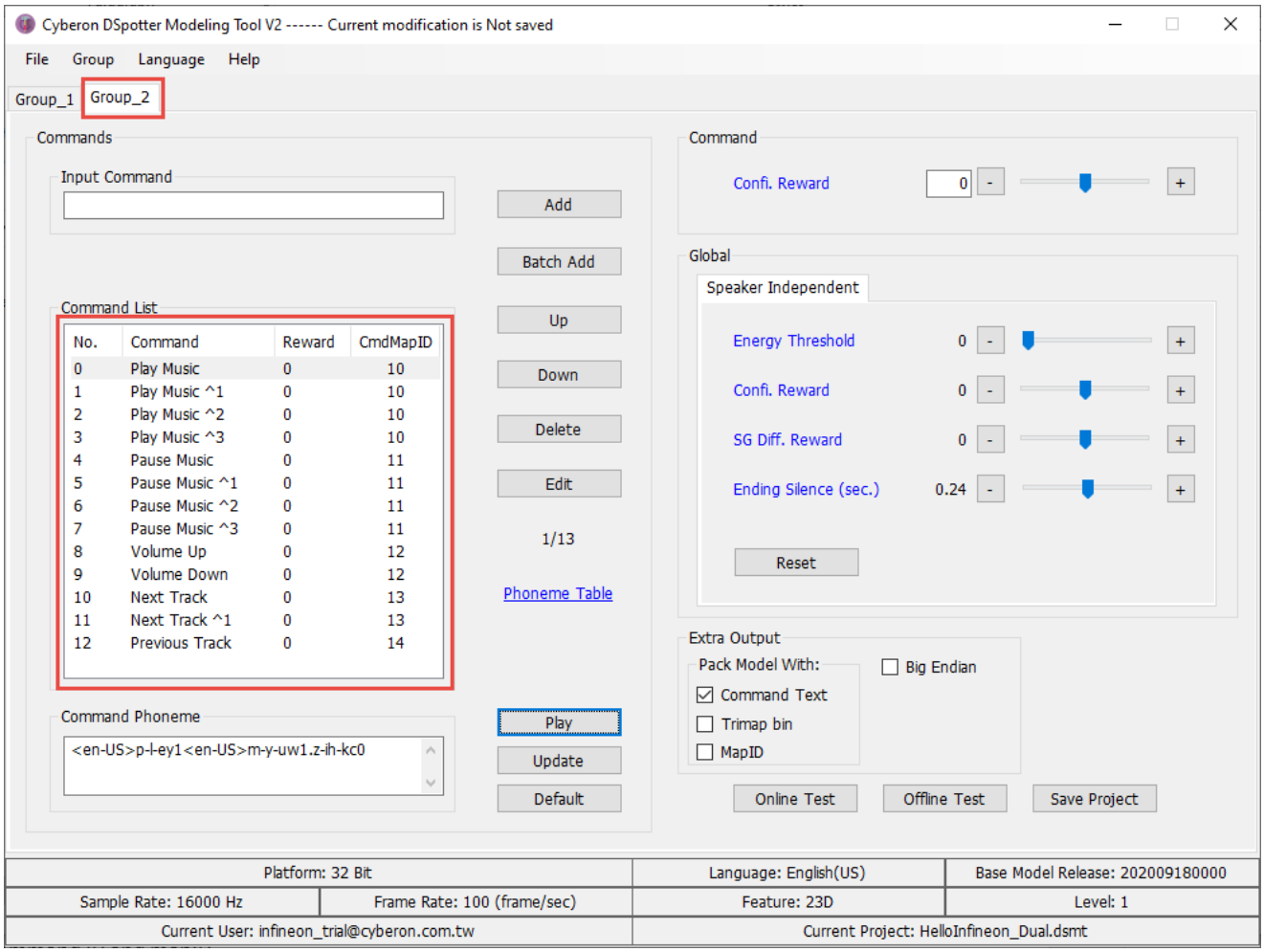
Go to the directory where you saved the DSMT project. In that directory, take the file called *<project_name>_pack_WithTxt.bin*. That file must be copied to the application's *data* directory and must be renamed to replace the existing file. For example, *data/One-Stage_pack_WithTxt.bin*.

Once you have done that, you must clean the application and then re-build. Otherwise it will not pick up the new model.

Note: *The file main.c has a `printf` that prints the trigger command. That is a hard-coded string so if you want it to match your trigger word, you must update it.*

5.2.7 Creating a two-stage custom model

To create a custom model for a two-stage keyword detection model, the process is the same, but you add a second group to hold the commands (**Group > Insert**). The trigger word stays on the **Group_1** tab while all sub-commands go on the **Group_2** tab. Choose a unique **CmdMapID** value for the trigger word and each command (use the same ID for different phonetic pronunciations of a given command) so that the callback function in the firmware will be able to tell you which command was issued.

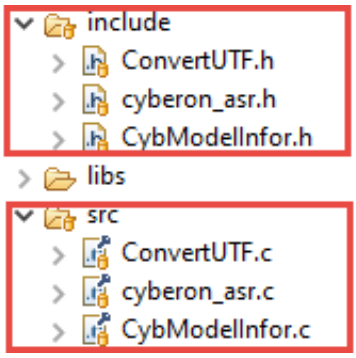


When you test a two-stage model, you will test each group individually. For example, when you test Group_2 as shown above, you don't have to say the trigger word (Hello Infineon) to test out the command keywords. However, on the device, the trigger word must be said before the command keywords for them to be recognized.

5.3 Firmware API

5.3.1 Cyberon library files

The Cyberon code is contained in the *include* and *src* directories inside the demo applications:



The file *src/cyberon_asr.c* is different between the single and two-stage demos because of the command timeout in the two-stage applications. All other files are the same.

5.3.2 User code

The user code is contained in *main.c* in all of the demo applications. The code is functionally equivalent for the one and two-stage solutions within a core type. That is, the one and two-stage demo applications for the CM4 are equivalent and the one and two-stage demo applications for the CM0+ are equivalent while the CM4 and CM0+ files are not the same. This is mainly because the CM0+ does not use the HAL or retarget-io libraries so PDL code is used for printing to the UART on the CM0+

Note: *The term "functionally equivalent" is used because the files are not identical but perform the same functions. They could (and most likely should) be identical.*

Note: *Only the CM4 version of main.c is described here. The CM0+ implements the same functionality without using the HAL or retarget-io library.*

5.3.2.1 Microphones

The `cyhal_pdm_pcm` HAL driver is used to collect data from the microphone and capture it in a buffer. When a frame is complete, the ISR callback function `pdm_pcm_isr_handler` reads the data alternately between two buffers called `pdm_pcm_ping` and `pdm_pcm_pong`. It also sets a flag variable called `pdm_pcm_flag` which is used to send the data to the keyword detection engine, as you will see in a minute.

5.3.2.2 Cyberon keyword detection engine

For the Cyberon keyword detection engine, you only need to include one header file, make two function calls, and provide one callback function. That's about as simple as it gets!

First is the header file:

```
#include "cyberon_asr.h"
```

Then in the initialization code, you call the Cyberon init function and provide it the name of your callback function:

```
/* Initialize Cyberon library */
cyberon_asr_init(asr_callback);
```

In the infinite loop, you wait for the flag that says you have a frame of audio data to process and then call the Cyberon function to process the data:

```
/* Process voice input */
if(pdm_pcm_flag)
{
   pdm_pcm_flag = 0;
   cyberon_asr_process(pdm_pcm_buffer, FRAME_SIZE);

}
```

The final step is to define the callback function. In the demo application, the callback just prints out the information it is passed:

```
void asr_callback(const char *function, char *message, char *parameter)
{
    printf("[%s]%s(%s)\r\n", function, message, parameter);
}
```

This callback is called multiple times for each recognized word, one for each type of message that is produced. The function is always `cyberon_asr_process`. The message is one of seven terms: (1) result delimiter string, (2) command name, (3) command ID, (4) map ID, (5) confidence score, (6) volume energy or (7) a timeout string for two-stage models. The parameter is the value for that particular message. For example, the result delimiter string is `++` for the start of a command and `--` for the end; the command name string is the ASCII name of the command that was spoken, and so on.

The confidence score and volume energy values are useful in debugging, but once you have a good model the Map ID is all you will really need – that tells you which command was issued. It will be the same value as the `CmdMapID` that you provided when you created the model.

As an example, here is what the output of a "Hello Infineon" command looks like:

```
[cyberon_asr_process]**********Result**********(++)
[cyberon_asr_process]Command(Hello Infineon)
[cyberon_asr_process]Command ID(1)
[cyberon_asr_process]Map ID(-1)
[cyberon_asr_process]Confidence Score(38)
[cyberon_asr_process]Volume Energy(6699)
[cyberon_asr_process]**************************(--)
```

The item in square brackets is the function, the middle item is the message, and the item in parenthesis is the parameter.

5.4 Exercises

Exercise 1: Create a custom one-stage keyword detection application

In this exercise, you will create the demo application for one-stage keyword detection running on a CM4. You will customize the trigger word and test out the performance.

1. Clone or download the `cyberon-one-stage-kwd-demo-psoc6-cm4` repo onto your local disk.

   Note: If you used the download option, unzip the file that was downloaded.

2. Create a new ModusToolbox™ application. On the application template page, use the **Browse…** button to navigate to the location where you downloaded the demo.
   Name your application **ch05_ex01_one_stage**.

3. Open a UART terminal window to your kit (baud 115200).

4. Build the application and program it to the kit. Note the `uniqueIdHi` and `uniqueIdLo` values for your kit.

5. Go to the Infineon DSpotter License website to get a Cyberon DSpotter license for your kit.

   Note: You can find a link to the website in the application's README.md file. The direct link is:
   https://license.cyberon.tw/InfineonDSpotterLicense/InfineonDSpotterLicense.php

6. Once the license file is emailed to you, save it and replace the file in *<app_path>/data/CybLicense.bin.*

7. Clean the application then rebuild it and program the kit.

8. Try saying "Hello Cyber Voice" to test the model's performance.

9. Open the Cyberon DSpotter Modelling Tool. Create a new project, and add a trigger word or phrase of your choice.

10. Use the Online Test feature to try out the model and adjust if necessary.

11. Save the model with **Pack Model With: Command Text** enabled.

12. Replace the model in the application with your new custom model.

13. Update the string that's printed in *main.c* to match your new trigger word.

14. Clean the application then rebuild it and program the kit.

15. Test out your new custom keyword.

Exercise 2: Create a custom two-stage keyword detection application

In this exercise, you will create the demo application for two-stage keyword detection running on a CM4. First you will customize the trigger word and commands, then test out the performance.

1. Clone or download the `cyberon-two-stage-kwd-demo-psoc6-cm4` repo onto your local disk.

Note: *If you used the download option, unzip the file that was downloaded.*

2. Create a new ModusToolbox™ application. On the application template page, use the **Browse…** button to navigate to the location where you downloaded the demo.
   Name your application **ch05_ex02_two_stage**.

3. Put the license file in the right place and program your kit to try out the default model.

Note: *The trigger word and commands can be found in the README.md of the application*

Note: *Remember that you can use the same license file as the prior exercise since you are using the same kit.*

4. Create a custom model to add Group_2 with a few commands.

Note: *You can copy the *.dsmt file from your existing DSMT project to a new directory, rename it, and then open it in DSMT as a starting point with your trigger word already done. You will need to add Group_2 with the second stage commands.*

5. Test your new commands using the online tester. Save the model when you are satisfied with it.

6. Include your custom model on the kit and test it.

Note: *Don't forget to fix the trigger word string in the `printf` in main.c*

Note: *Remember to clean the application and rebuild it after replacing the model file.*

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2022 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.