ON Semiconductor

Is Now

To learn more about onsemi™, please visit our website at
www.onsemi.com

AND9267/D

Getting Started with BelaSigna® R281

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

Introduction

This application note guides you through the process of integrating BelaSigna R281 into a new prototype or design. It will first discuss the BelaSigna R281 voice trigger algorithm in terms of functionality. Next, the main points to consider as part of the design−in process are discussed, and two reference circuit diagrams are presented. Finally, configuration and operation of BelaSigna R281 via its I^2C control interface is discussed.

The intended audience is system architects and engineers designing products that can benefit from the always−on voice trigger functionality provided by BelaSigna R281.

Always−On Voice Trigger

Most smartphones today contain extremely capable speech recognition engines, allowing you to issue voice commands to your device without ever having to type anything in. However, these speech recognition engines are typically only active when you physically press a button or launch an application, as they consume a significant amount of power listening for and recognizing speech. If they were active all of the time, the resulting decrease in your device's battery life would be totally unacceptable.

BelaSigna R281 is an ultra−low−power, always−on, always−listening voice trigger device. Voice trigger is useful as it provides an always−active, extremely low power means of waking up an electronic device such as a smartphone or tablet, without having to physically touch the device. Essentially, it replaces a physical pushbutton with a voice−activated switch. Being able to wake up your smartphone completely hands−free has many advantages, not the least of which is safety. It is much safer to be able to speak naturally to your phone while driving than it is to manipulate a user−interface with your hands.

Algorithm Details

The algorithm executing inside BelaSigna R281 consists of an ultra−low−power voice activity detector, combined with an audio waveform pattern matcher. The voice activity detector is constantly executing and listening for what it considers to be speech, captured on either an analog or digital microphone input (separate firmware images are available for an analog versus a digital microphone). Once speech is detected, frames of data are analyzed and specific features are calculated and buffered. These features are compared to sets of features that were calculated during the training process, and if they are considered similar enough to any of the training templates a match is declared and indicated on the wake−up pin.

Because the algorithm inside BelaSigna R281 is a waveform pattern matcher and not a general speech recognition engine, it must be given a reference pattern to match. It uses more than one instance of a reference pattern to account for slight differences in how a typical user will say a given phrase to further improve performance. These patterns are obtained during a training phase, and consequently BelaSigna R281 must be trained before it can be placed into Recognition Mode. These reference patterns calculated during the training phase are known as *training templates*.

Having a pattern−matching algorithm that is not a general speech recognition engine has some advantages. In particular, it is language independent and there is nothing specific that ties the algorithm to matching speech. It can just as easily match audio signals that are not speech as well. Finally, it is a relatively straightforward implementation algorithm−wise that does not require large amounts of memory, allowing for a small−footprint, deeply−embedded implementation that consumes very little power, allowing it to be always−on and always−listening.

Initial Power−On State

When BelaSigna R281 is powered on, the device will perform a brief initialization procedure and then wait for a connection to be made from an external host via I^2C. At this point, the host controller must connect to BelaSigna R281 and load its memory with the desired algorithm binary image, as well as the training template data. Once this has been completed, the device can be put into Recognition Mode. If no training template data is available (e.g. the training procedure has never been performed), then BelaSigna R281 must be placed into Training Mode and the

training procedure performed before entering Recognition Mode.

Whenever power is removed from the device, the contents of memory are lost and must be reloaded. For more information on loading an appropriate firmware image over I^2C, refer to *Booting an Application via I^2C*. For details on switching modes, and saving and loading training templates refer to *Controlling the Algorithm*.

Algorithm Modes

Once the base firmware image has been loaded into RAM and the main algorithm is running, there are four main modes possible with BelaSigna R281: Sleep Mode, Standby Mode, Training Mode, and Recognition Mode.

For details on switching modes, refer to *Switching Modes*.

Sleep Mode

In this mode BelaSigna R281 is in a low−power state with all inputs disabled and no audio is being collected or processed. The only thing that is active is the I^2C port, allowing configuration and control of the device. Sleep Mode consumes the least current of all of the modes.

Standby Mode

The algorithm initially starts up in Standby Mode. In this mode BelaSigna R281 is in a low−power state with the microphone enabled. Audio is being collected and processed, but only signal statistics are being collected and no recognition data is buffered. The I^2C port is also active, allowing configuration and control of the device. Standby Mode consumes very little power; slightly more than Sleep Mode and less than Recognition or Training Mode.

While it is perfectly valid to issue I^2C commands in either Training Mode or Recognition Mode, it is recommended that the device be placed into Standby Mode before issuing any I^2C commands that will affect operation of the device (e.g. saving or loading training templates, changing preamplifier gain or microphone bias level, or changing the wake−up pin configuration).

Training Mode

Training BelaSigna R281 to recognize your specific trigger phrase is performed in Training Mode. In this mode BelaSigna R281 enables the microphone (analog or digital, depending on the firmware loaded onto the device), and waits for an audio input signal loud enough to trigger the voice activity detector. Once audio is detected, frames of audio are analyzed and features are buffered until the input signal stops, or the recording buffer is full (the buffer can hold approximately 1.5 seconds of audio). Only a single utterance is captured. Once the end of input is detected, the device will automatically switch back to Standby Mode.

A separate parameter controls the active training template. There are three templates in all: Template 0, Template 1, and Template 2. When in Training Mode, the currently selected template is overwritten. Template 0 is unique in that when Template 0 is the active template, whenever the device transitions from any mode into Training Mode, all templates are erased and any training results are effectively cleared.

To perform a complete training using Training Mode, the order of events is as follows:

1. Place the device in Standby Mode
2. Set the active template to Template 0
3. Place the device in Training Mode
4. Wait for the training iteration to complete by polling the current mode. When the device switches to Standby Mode, the current training template was captured.
5. Update the active template to Template 1
6. Place the device in Training Mode
7. Wait for the training iteration to complete by polling the current mode. When the device switches to Standby Mode, the current training template was captured.
8. Update the active template to Template 2
9. Place the device in Training Mode
10. Wait for the training iteration to complete by polling the current mode. When the device switches to Standby Mode, the current training template was captured.

At this point, all three training templates should be captured and will be present in RAM and the training results can be validated. If the training results are valid the device can be placed into Recognition Mode. Refer to *Validating Training Results* for more information on validating the training results.

Power consumption is highest in Training Mode as this mode is rarely used, and is only used when the main external host is awake anyway. As a result the system clock rate is set at the maximum value at all times while in Training Mode to provide the most computational power.

Once the training procedure has been completed, the three training templates can be read out of the device and saved offline to be loaded directly into RAM in the future, avoiding the need to re−run the training process. For additional details on saving and loading training templates refer to *Saving/Restoring Training Template Data*.

Recognition Mode

When a complete set of three valid training templates is present in memory, BelaSigna R281 can be placed into Recognition Mode. If you attempt to place BelaSigna R281 into Recognition Mode without a valid set of templates in memory, the command to switch modes will be ignored.

When the device is in Recognition Mode, the microphone (analog or digital, depending on the firmware loaded onto the device) is enabled and BelaSigna R281 waits for an audio input signal loud enough to trigger the voice activity detector. Once audio is detected, frames of audio are analyzed and features are buffered until the input signal stops, or the recording buffer is full (the buffer can hold approximately 1.5 seconds of audio). When the end of input is detected the current contents of the recording buffer are compared to each of the three training templates and a

similarity measurement is determined. If this similarity measurement is below a certain threshold (defined by the *Decision Threshold* parameter) a match is declared and this status is indicated on the wake−up pin. Otherwise, the contents of the recording buffer are discarded and the device goes back to waiting for audio input.

Power consumption in Recognition Mode is, on average, extremely low. The voice activity detector and feature computation takes a relatively small amount of computational power and is performed at an extremely low system clock rate. The system clock is only increased during the computationally−intensive similarity measurement stage. While power consumption during this time period is noticeably higher, the amount of time spent in this calculation is quite small. Thus, on average, the power consumption in Recognition Mode is optimized to be quite low (less than 300 uW for a supply voltage of 1.8 V).

The behavior of the wake−up pin is configurable via the *Wake Pin Configuration* parameter and is described in detail in *Wake−Up Pin Configuration*.

The Design−In Process

When designing BelaSigna R281 into a product, there are a few things to consider upfront that can help to optimize your design in terms of power consumption and overall design complexity. This section discusses these items, and presents a reference circuit illustrating the various options.

Microphone Selection

BelaSigna R281 will accept either an analog or a digital microphone input. A separate firmware image is available providing support for either one of these microphone options so ensure you are using the correct firmware build based on your selection of microphone technology.

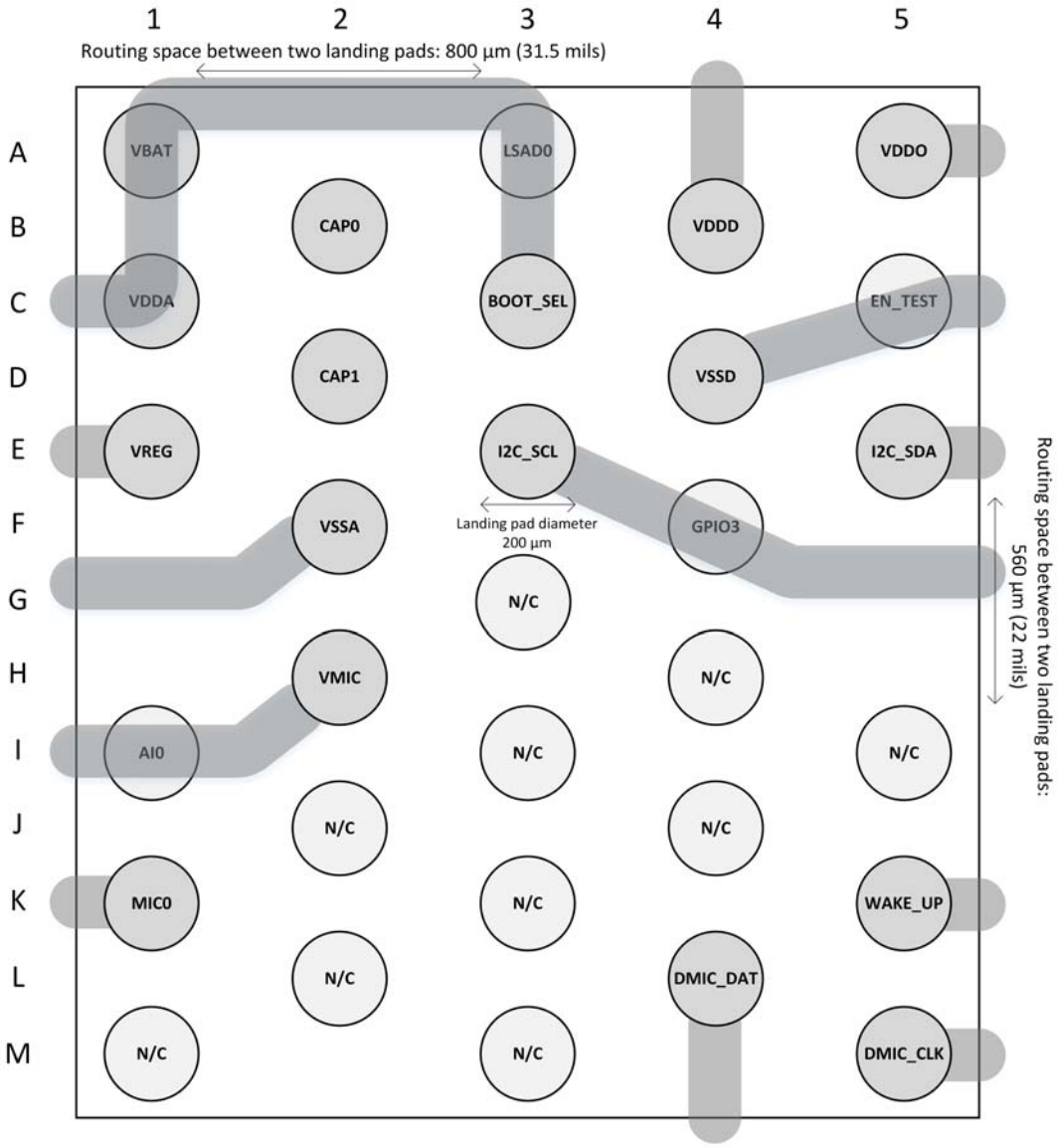
An analog microphone is recommended as typically they consume significantly less power than a digital microphone. Both options are discussed in *Reference Circuit*.

Power Supply Considerations

The BelaSigna R281 must be powered using a supply voltage of 1.8 V.

PCB

It is possible to route the WLCSP package on a single layer using 5 mil trace and space design rules. This requires some signals to be routed through unused balls. The WLCSP reference schematic shown in Figure 2 and the suggested PCB routing diagram shown in Figure 1 illustrate exactly how this can be achieved.



BelaSigna R281 WLCSP 5 mil routing - top and soldering footprint view
(balls facing down)

Figure 1. BelaSigna R281 WLCSP Suggested PCB Routing

Reference Circuit

The reference circuit for BelaSigna R281, shown in Figure 2, is the simplest to design in and will result in the lowest possible power consumption as the charge pump is disabled. The schematic shows options for both an analog (MEMs) or digital microphone. Choose whichever microphone you prefer, keeping in mind that analog microphones typically consume less power than digital microphones. The analog microphone power supply (VMIC) is an output from BelaSigna R281 and can be configured to either 1 V or 2 V over I^2C via the *VMIC Configuration* parameter. If you are using a two−terminal electret microphone, ensure you use an appropriate bias resistor between the microphone signal and VMIC (2.2 kΩ is recommended).

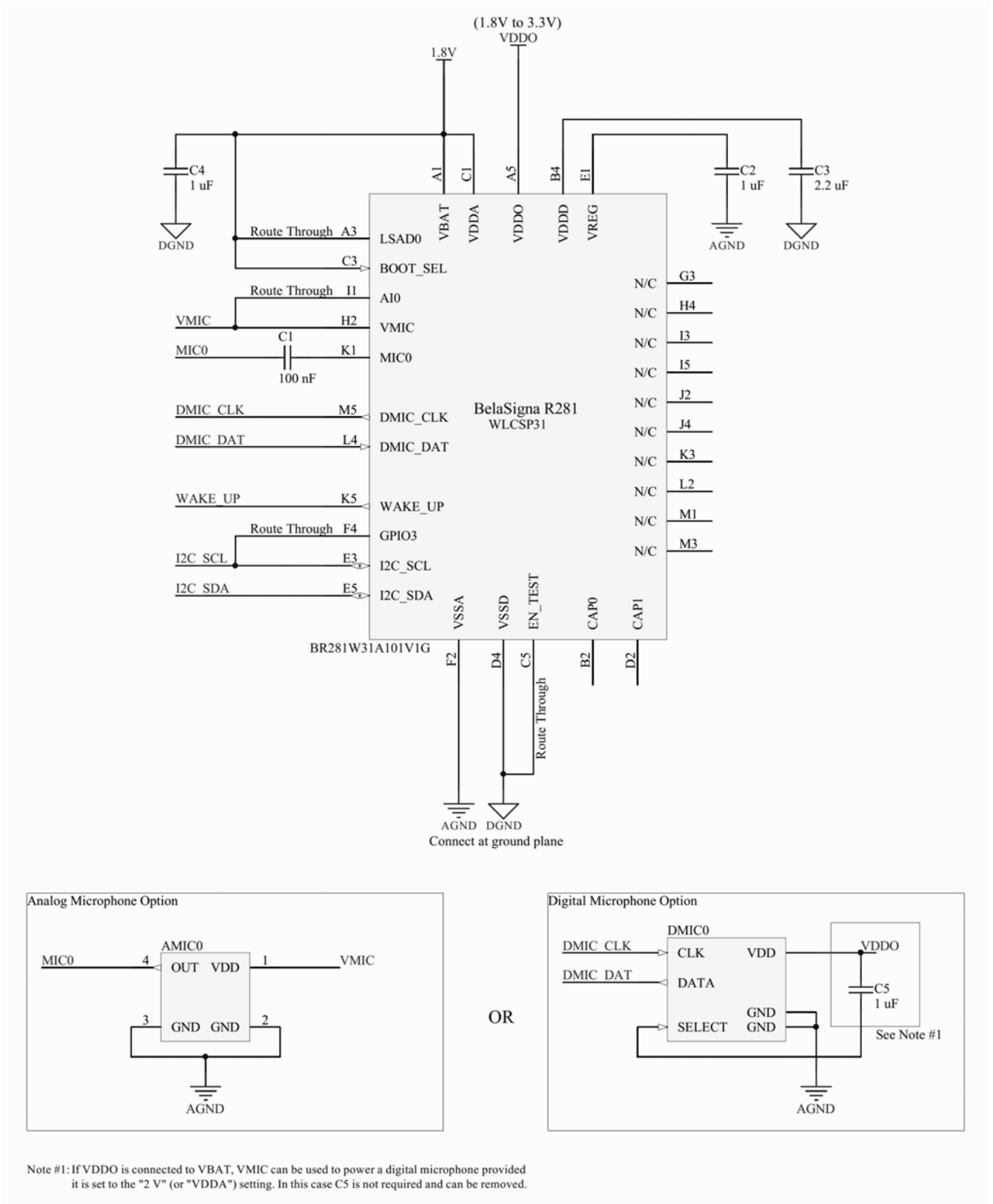


Figure 2. BelaSigna R281 System Diagram (WLCSP Package)

Preamplifier gain is adjustable from 0 dB to 30 dB in 3 dB steps via the *Preamp Configuration* parameter. The default preamplifier gain setting of 18 dB is appropriate for a microphone with a sensitivity of −42 dB (where 0 dB = 1 V/Pa, @ 1 kHz). For more information on hardware−related parameters refer to *Configuration of the Hardware*.

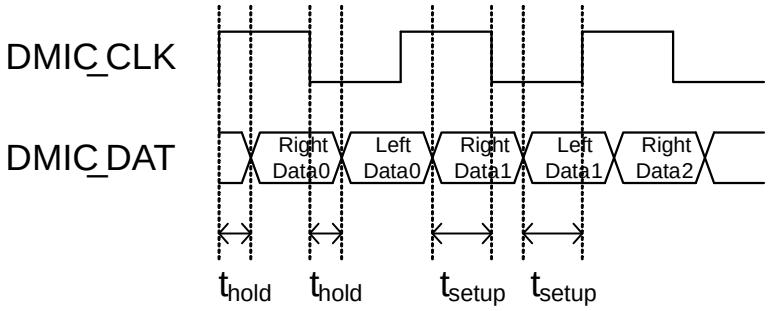


Figure 3. DMIC Timing Diagram

I²C Command Protocol

BelaSigna R281 features an extensive I²C command interface that allows an external I²C master device to configure the hardware and various elements of the voice trigger algorithm.

The I²C interface on BelaSigna R281 always operates in slave mode at a fixed, 7−bit slave address of 0x62. The maximum I²C clock speed is 100 Kbps.

Data transfers using the I²C interface use 8 bits of data in every frame sent. The first frame in each data transfer must be the address of the device with which the master wants to communicate, followed by a single bit in the least significant bit (LSB) position indicating whether a read or a write will be performed by the master (set to 0 for writing or 1 for reading).

The BelaSigna R281 I²C protocol is split into two sections:

- Device control and programming providing low−level access to the system
- Algorithm configuration and control commands

The lower−level device control and programming commands provide a means of bootstrapping the system and loading the voice trigger firmware itself, while the loaded firmware defines the remainder of the I²C command set related to algorithm configuration and control.

Device Control and Programming

The low−level device control and programming command set is summarized in Table 1 through Table 11.

Table 1. I²C NOP COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
NOP	0x00	None	None

This command performs no operation with no side effects.

Table 2. GET CHIP ID COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Get Chip ID	0x56 (‘V’)	None	High byte Low byte

This command returns the BelaSigna R281 chip identifier. The chip identifier should have the value 0x5000.

Table 3. SET MEMORY BLOCK POINTER COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Set Memory Block Pointer	0x4D (‘M’)	Address high byte Address low byte Size high byte Size low byte Memory space byte	None

This command sets the internal memory block pointer to the specified address and size. The internal memory block pointer is used for the *Write Memory*, *Read Memory* and *Calculate Checksum* commands. This command allows you to specify a block of the memory spaces to write to, read from or calculate a checksum for. To fully define the expected range of the memory access commands that are expected to follow, the parameters associated with the command specify:

- The memory space
- Where memory accesses will start
- How many words of memory will be processed

This command also initializes the checksum to the sum of the memory address pointer and the memory space enumeration. Once you set the memory block pointer, it remains set until you set it again, set the *Current Template* parameter, or reset the chip.

The memory space enumeration value specifies which of the P, X or Y memory spaces contains the memory block of interest. The possible enumeration values are:

- 0x0 – for X memory
- 0x2 – for Y memory
- 0x4 – for P memory

The address bytes are the high and low bytes of the word−length address that specifies where in the specified memory space the follow−up associated memory access command will start processing.

The high and low size bytes specify the word−length size parameter, which indicates how many words of data are used in the follow−up memory access.

For example, to set the memory block pointer to Y:0x2048 for 0x1FF words, send the following frame:

‘M’ 0x20 0x48 0x01 0xFF 0x2

Table 4. WRITE MEMORY COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Write Memory	0x57 (‘W’)	High byte, low byte ... High byte, low byte	Checksum high byte Checksum low byte

This command writes to the memory block pointed to by the internal memory block pointer. The memory block pointer specifies the memory space and starting address to write the data input to, and the number of data byte pairs to expect. If an odd number of bytes is received, the last byte is ignored.

While receiving data, the system continuously calculates a checksum for the data received. At any point following data transmission, you can use the *Get Checksum* command to get the checksum of the data that has been written so far. Once the data transmission has completed, the *Get Checksum* command is automatically executed to reduce the communications overhead related to downloading and verifying data transmissions.

If the internal memory block pointer was not previously defined through the *Set Memory Block Pointer* command, behavior of this command is undefined.

Table 5. READ MEMORY COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Read Memory	0x52 (‘R’)	None	High byte, low byte ... High byte, low byte

This command reads from the memory block pointed to by the internal memory block pointer. The memory block pointer specifies the memory space and starting address from which to read the data input, and the number of data byte pairs to expect.

If the internal memory block pointer was not previously defined through the *Set Memory Block Pointer* command, behavior of this command is undefined.

Table 6. CALCULATE CHECKSUM COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Calculate Checksum	0x4B (‘K’)	None	Checksum high byte Checksum low byte

This command calculates the checksum for the data in the memory block pointed to by the internal memory block pointer. The memory block pointer specifies the memory space and address at which to start, and number of words to include in the checksum calculation. The checksum is a 16−bit truncated sum of all of the data in the block, plus the memory address pointer and memory space enumeration specified by the memory block pointer. Once the calculation has completed, the *Get Checksum* command is executed automatically to reduce communications overhead. If the internal memory block pointer was not previously defined through the *Set Memory Block Pointer* command, behavior of this command is undefined.

Table 7. GET CHECKSUM COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Get Checksum	0x43 (‘C’)	None	Checksum high byte Checksum low byte

This command returns the most recently calculated checksum, typically corresponding to the memory space pointed to by the internal memory block pointer. After this command, multiple I^2C reads of the interface repeatedly return the calculated checksum.

This command is automatically executed following the *Write Memory* and *Calculate Checksum* commands.

Table 8. START APPLICATION COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Start Application	0x47 (‘G’)	Address high byte Address low byte	None

This command starts execution of the DSP at the specified address. You can use this command to start an application that was downloaded to BelaSigna R281 through the *Write Memory* command. Execution starts at the word length address specified by the high and low bytes transmitted as parameters to this command.

When executing the application, the built−in I^2C protocol command handler is exited.

Table 9. READ EXT3 COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Read EXT3	0x44 (‘D’)	None	High byte Low byte

This command returns the contents of the EXT3 register.

Table 10. WRITE EXT3 COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Write EXT3	0x45 (‘E’)	High byte Low byte	None

This command writes a value to the EXT3 register.

Table 11. EXECUTE JumpROM COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Execute JumpROM Function	0x4A (‘J’)	JumpROM function byte	None

The *Execute Jump ROM* command causes the JumpROM function specified in the input byte to be executed. The JumpROM function set provides access to certain low−level functions in the Program ROM. These functions allow you to get the Program ROM version and reset the system. You can call the JumpROM functions through the built−in I^{2}C protocol. When running this command, control of the device is transferred to the JumpROM function until the JumpROM function completes. As a result, the system is unresponsive and does not acknowledge any further I^{2}C commands until the JumpROM function has completed. Provided that the master device communicating with BelaSigna R281 does not experience difficulties following a series of non−acknowledged bytes, you can test for the completion of the JumpROM function by polling the interface using the NOP command (specified in Table 1) until BelaSigna R281 acknowledges the I^{2}C transmission again. Any invalid and undefined functions requested from the JumpROM function selector are treated as though they were NOP function requests.

The following JumpROM functions are available:

Table 12. JumpROM FUNCTIONS

Function Name	Function Byte	Description
NOP	0x00	No operation
Get Version	0x01	Fills EXT3 with the Program ROM version. The contents of EXT3 can be read with the *Read EXT3* command.
Reset System	0x02	Forces a system reset

Booting an Application via I^{2}C

The device control and programming commands previously described can be used to bootload BelaSigna R281 with a specific set of firmware over the I^{2}C interface. Once an application resides in the program RAM and the X and Y data memories, it can be executed with the *Start Application* command.

By default, BelaSigna R281 boots into a “wait for I^{2}C host” state and after a short period of time will reset if no host attaches. Thus, it is recommended that you put the system into a known state before initiating the application download. The best way to achieve this is by issuing the JumpROM NOP command, which causes the system to re−initialize and wait in the JumpROM wait loop until another I^{2}C command is received or the system is reset. After booting the chip, the NOP command can be continually sent until an ACK is received from BelaSigna R281. Receiving an ACK means that BelaSigna R281 has successfully responded to the NOP command, and is ready for further communications.

At this point the I^{2}C interface is operating properly, and the system is ready to start the process of downloading the object code to the three different memory spaces (X, Y and P) of BelaSigna R281.

The firmware for BelaSigna R281 is available in various file formats from ON Semiconductor, one of which is a C−Header file (as described later in this document), making it extremely simple to automate the process of downloading the object code into the memories of BelaSigna R281 from a C program using the aforementioned low−level I^{2}C commands. Sample code in the C programming language implementing the entire I^{2}C command set and automating the bootloading process is available from ON Semiconductor. This sample code is discussed in detail in *C Sample Code*.

Algorithm Configuration and Control

Once the core voice trigger application has been loaded onto the device, there are three additional commands available via I^{2}C: the *Reset Application*, *Set Parameter Register* and *Get Parameter Register* commands. These commands are described in Table 13 through Table 15.

Table 18 contains a list of all of the available algorithm parameters that can be read and written via I^{2}C. For your convenience, the C−based sample code provides higher−level function wrappers around many of these parameters. Refer to *C Sample Code* for more information.

Table 13. RESET APPLICATION COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Reset Application	0x01	None	None

This command resets the program counter to entry point of the application and executes the code from that point.

Table 14. SET PARAMETER REGISTER COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Set Parameter Register	0x32	Parameter number Parameter value high−byte Parameter value low−byte	None

This command updates the specified algorithm parameter number with the parameter value passed in.

The two−byte parameter value word must be sent MSB−first. See Table 18 for a complete list of available parameters.

Table 15. GET PARAMETER REGISTER COMMAND

Command Description	Command Byte	Input Bytes	Output Bytes
Get Parameter Register	0x33	Parameter number	Parameter value high−byte Parameter value low−byte

This command returns the specified algorithm parameter's value. The two−byte parameter value is returned MSB−first.

See Table 18 for a complete list of available parameters.

C Sample Code

To make the design−in process as simple as possible, ON Semiconductor provides a comprehensive sample application written in the C programming language that not only provides a high−level wrapper around the entire I^2C protocol, but also shows you how to properly initialize and connect to BelaSigna R281 and load the firmware. It also provides functions that show you how to properly switch modes, perform the training process, validate the training results and save and restore the training templates.

This section will highlight the key features of this sample code, and show you how to integrate it into your own code base.

Note that the C programming language was chosen as this is the expected programming language for most embedded systems. Should you require examples in other programming languages, please contact your local ON Semiconductor support representative.

The remainder of this section discusses the C sample code in more detail and assumes a minimum knowledge of embedded programming and C. Should you have any questions, please contact your local ON Semiconductor support representative.

BelaSigna R281 Firmware Variants

Looking at the sample code, you will see four different variants of the BelaSigna R281 firmware. Which one you choose depends on whether you have a supply voltage of 1.8 V or higher, and whether or not you are using an analog or a digital microphone. The sample code has defines at the top of `BelaSignaR281_sample.cpp` that selects the correct firmware include file based on the selection of supply voltage and microphone type (via the #defines `SUPPLY_VOLTAGE` and `MIC_SELECTION`). Adjust these defines to correspond to your specific design, and the correct firmware version should automatically be included.

Structure of the C Sample Code

The C sample code is broken into a series of files that separates the generic I^2C protocol from any machine−specific communications code. The sample also includes a Windows® command line program that can be used to exercise the API, and this is also isolated into a separate file. The files included in the C sample along with their specific functions are summarized in Table 16 and Table 17.

Table 16. C SAMPLE CODE HEADER FILES

Filename	Description
general_defs.h, stdafx.h, targetver.h	Windows−specific include files. Not required for a typical non−Windows application.
comm.h, util.h	Include files declaring a hardware−agnostic I^2C communications API and various utility functions
i2c_commands.h	Include file describing the BelaSigna R281 I^2C command protocol and higher−level utility functions related to it
BelaSignaR281_fw_*.h	Various firmware images for BelaSigna R281
wilhelm.h	Include file containing a fixed set of training templates (for illustrative purposes)

Table 17. C SAMPLE CODE SOURCE FILES

Filename	Description
stdafx.cpp	Windows−specific implementation file. Not required for a typical non−Windows application.
util.cpp	Windows implementation of the utility functions related to delays and timing
comm.cpp	Implementation of the generic, low−level I^{2}C communications API supporting the ON Semiconductor Communications Accelerator Adaptor, Total Phase Aardvark and Total Phase Promira device (through the ON Semiconductor Communications Toolkit). You must provide an equivalent implementation for your specific I^{2}C master device.
i2c_commands.c	Hardware−agnostic implementation of the BelaSigna R281 I^{2}C command protocol and higher−level utility functions related to it
BelaSignaR281_sample.cpp	An example Windows console application that uses the BelaSigna R281 I^{2}C protocol to perform various tasks. Use this as a reference when implementing your own BelaSigna R281 driver.

Integrating the C Sample into Your Code

The intent of the C sample code is to provide as complete an example as possible to make it easy to integrate into your code. When porting this code to your specific embedded system, it should be a simple matter of:

1. Removing the unnecessary, Windows−specific framework files (`general_defs.h`, `stdafx.h`, `targetver.h`, `stdafx.cpp`, and `BelaSignaR281_sample.cpp`)
2. Providing implementations of the low−level I^{2}C communications functions found in `comm.cpp` and delay− and timing−related functions in `util.cpp` specific to your hardware
3. Compiling for your specific architecture

Once those steps are complete, you can use the BelaSigna R281 I^{2}C protocol in your own driver to perform the various functions you need in your application, using the example Windows console application as a reference.

Important Functions

The Windows console application has a number of functions that are key to using BelaSigna R281 effectively.

Initial Connection to BelaSigna R281

The initial connection to a freshly−booted BelaSigna R281 chip is important, and the `connect()` function in `BelaSignaR281_sample.cpp` combined with the `I2CConfirm(timeout_ms)` function in `i2c_commands.c` illustrates how to do this reliably.

Loading Firmware onto BelaSigna R281

Once you have connected to BelaSigna R281 via its I^{2}C port, you must load a version of the firmware into its RAM and execute the algorithm. This is illustrated in the `initialize()` function found in `BelaSignaR281_sample.cpp` which calls the `DownloadFirmware(datablocks[], length)` and `Run()` functions in `i2c_commands.c`.

The firmware to download is selected via defines at the top of `BelaSignaR281_sample.cpp`. Make sure you select the appropriate firmware image for your design.

At this point the algorithm will be executing, and you can use the rest of the I^{2}C commands and higher−level functions to configure and control the device. Some of the more common tasks have been wrapped in functions for convenience, and these are discussed in more detail in the next section.

Controlling the Algorithm

There are a few main tasks that you will perform with BelaSigna R281 to implement always−listening voice trigger:

- Switching modes
- Training
- Validating training results
- Saving/restoring training template data
- Adjusting the decision threshold
- Configuration of the hardware (e.g. wake−up pin, microphone bias, preamplifier gain)

Switching Modes

Mode switching is accomplished by setting the *Mode* parameter. There are four modes: Sleep Mode, Standby Mode, Training Mode, and Recognition Mode. Changing modes can be done at any time, and is as simple as writing the correct mode number into the *Mode* parameter.

There is a `SetMode(mode)` function provided for you in `i2c_commands.c` that sets the *Mode* parameter, and verifies that the mode switch occurred. It is recommended that you use an approach similar to this when setting the mode.

Note that you will not be able to switch into Recognition Mode if the device has not been trained.

Training

Training is performed as discussed in Training Mode. The procedure outlined in the aforementioned section is wrapped up in a function called `train_template(template_number)` found in `BelaSignaR281_sample.cpp`.

When this function is called with template number 0 as the argument, all templates are erased and the first training template is recorded. You would then call the function again with template 1 as the argument, and a third time with template 2 as the argument. After the third template is recorded, the device will automatically call `RecalculateTemplateStatus()` to update the

Template Status parameter, and revert back to Standby Mode. If the *Template Status* word indicates a successful training, the device can be placed into Recognition Mode. However, it is usually desirable for the external host to perform some offline validation of the training results at this time.

One other restriction that is highly recommended is some form of noise floor check to ensure that training cannot be performed in a noisy environment. This can be as simple as getting the *Noise Floor* or *Short Term Energy* parameter value (which is a 32−bit value, and must be retrieved with two separate calls to *Get Parameter Register*), and preventing the user from initiating training when the noise floor is above a certain threshold. This threshold will depend on your particular design, microphone type, and preamplifier gain and should be determined through testing. Note that the noise floor is not updated when BelaSigna R281 is in Sleep Mode (since the input stage is disabled), so ensure the device is in Standby Mode when reading these signal statistics. The functions `GetNoiseFloor()`, `GetFrameEnergy()` and `GetShortTermEnergy()` in `i2c_commands.c` all return a full, 32−bit value.

Validating Training Results

One you have captured three training templates, there are some basic sanity checks that can be done to ensure the training procedure went smoothly, and that the results in Recognition Mode will be reliable. For example, it is possible that the user did not say the same phrase every time, or that they attempted to train the device in a noisy environment, both of which could result in a bad set of training templates and poor performance in Recognition Mode. BelaSigna R281 simply validates that the training templates exist in memory and that they have a non−zero length, and that the decision threshold is non−zero. It does not perform detailed validation of the training results as this can be easily done offline by the external host, and provides more flexibility for your particular design.

The `is_training_valid()` function found in `BelaSignaR281_sample.cpp` shows you one example of how you could validate the training results. It starts by validating that the three training templates are all roughly the same length (in number of frames, where each frame represents 16 ms of audio). The template lengths are available in the *Template N Length* parameters (where ‘N’ is 0, 1 or 2). While it is expected that the training templates will differ in length (indeed part of the algorithm specifically compensates for this), we can make an assumption that if the length difference is too much, the training is invalid. An absolute value of 30 frames is selected as the limit (or approximately 500 ms), but this value can be whatever you like, and is best validated through testing.

Second, the `is_training_valid()` function retrieves the three intra−template distances (the *Intra−Template Distance N* parameters; where ‘N’ is 0, 1, or 2) and determines if any of these three distances looks out of place. If the three training templates were all identical, the intra−template distances would all be 0. In reality, these three numbers should be relatively close to one another (within approximately 100). The `is_training_valid()` function checks if any intra−template distance is too large with respect to the others and will fail if it is above a maximum threshold. Alternatively, it will issue a warning if any intra−template distance is above a slightly lower threshold, which could be used to warn the user performing the training that the results are not ideal, and that they may want to consider performing the training again for best results.

Ultimately, it is up to you how you want to validate the training results. The sample code gives you one example, but you may want to employ your own heuristic using the data available to you via the algorithm parameters listed in Table 18, and some testing of your device under various conditions.

Saving/Restoring Training Template Data

Once the training process is complete, the training templates will need to be read out of the memory of BelaSigna R281 and saved offline (BelaSigna R281 does not contain any non−volatile memory, and the training templates will be lost when power is removed from the device). The *Current Template* parameter (which calls the *Set Memory Block Pointer* command when it is written) combined with the *Read Memory* and *Write Memory* I^2C functions can be used to read and write the training template memory areas.

To make things extremely easy, two higher−level functions are provided in `i2c_commands.c`: `ReadTemplateData(template_num, *data)`, and `WriteTemplateData(template_num, *data)`. These functions take a template number and a pointer to an array of bytes to read from or write into, and they take care of all the machinery of switching to Standby Mode, setting the active template number calling *Read Memory* or *Write Memory*, and restoring the previous mode for you. All you need to do is persist the training data (the array of bytes read from BelaSigna R281) to a file or some other non−volatile area for re−loading later. You will need to call the `ReadTemplateData` or `WriteTemplateData` function three times (once for each training template) to read or write all of the training data. Note that `ReadTemplateData` will fail if you try to read an empty training template (one that has not been trained).

Reading training template data is fairly straightforward. However, when loading training template data directly into the memory of BelaSigna R281 with the `WriteTemplateData` function you also need to manually trigger a recalculation of the *Template Status* word as well as the intra−template distances and decision threshold. This is accomplished by calling the `RecalculateTemplateStatus()` function in `i2c_commands.c`. Assuming the training template data

you are loading has previously been validated using your own heuristic, you can now place the device into Recognition Mode.

In summary, to avoid having to re−train the device every time the power is cycled, the training templates should be read out of memory and saved offline using the `ReadTemplateData` function. These saved training templates can then be loaded directly into memory (immediately after the BelaSigna R281 firmware is loaded) when power is restored to the device.

The correct procedure for reading out all three training templates is:

- Put the device into Standby Mode
- Call `ReadTemplateData(0, *data0)`
- Call `ReadTemplateData(1, *data1)`
- Call `ReadTemplateData(2, *data2)`
- Save `data0`, `data1`, and `data2` offline in non−volatile memory

To re−load the saved training template data into memory and start BelaSigna R281 directly in Recognition Mode:

- Put the device into Standby Mode
- Read the training templates out of non−volatile memory into byte arrays (e.g. `data0`, `data1`, and `data2`)
- Call `WriteTemplateData(0, *data0)`
- Call `WriteTemplateData(1, *data1)`
- Call `WriteTemplateData(2, *data2)`
- Call `RecalculateTemplateStatus()`
- Put the device into Recognition Mode

This entire procedure is illustrated in `BelaSignaR281_sample.cpp` in the snippet of code that handles the `load_wilhelm` command line option. This command line switch loads a hard−coded set of training templates that are set to match the Wilhelm Scream sound clip, which is freely available and can be found at https://archive.org/details/WilhelmScreamSample.

Adjusting the Decision Threshold

One of the most important algorithm parameters is the *Decision Threshold*. This value is typically calculated by the algorithm during the training phase and can be read out at any time by reading the *Decision Threshold* parameter. This parameter is also writable, and as a result you can adjust the tolerance of the algorithm's matching engine by adjusting this parameter up or down.

The default value calculated by the algorithm is chosen to be a good balance between missed triggers and false acceptances. Decreasing this value will make the matching algorithm more restrictive (increasing the potential for missed triggers), and increasing this value will cause the algorithm to match more loosely (increased false triggers).

Should you decide to change this parameter, it is up to you to rigorously test your system to ensure acceptable performance.

Configuration of the Hardware

There are a few parameters related to the configuration of the hardware itself that are likely to be adjusted based on your device design.

Preamplifier Gain

The analog microphone preamplifier gain is adjustable using the *Preamp Configuration* parameter. Valid values for this parameter are as follows:

0	0 dB
1	3 dB
2	6 dB
3	9 dB
4	12 dB
5	15 dB
6	18 dB
7	21 dB
8	24 dB
9	27 dB
10	30 dB

The default preamplifier gain setting of 18 dB is appropriate for a microphone with a sensitivity of −42 dB (where 0 dB = 1 V/Pa, @ 1 KHz).

Microphone Bias/Supply Pin

The level of the VMIC output pin is adjustable using the *VMIC Configuration* parameter. Valid values for this parameter are as follows:

0	VSSA (analog ground)
1	VREG (1 V)
2	VDDA (2 V, unless connected to VBAT)
3	HI−Z

Wake−Up Pin Configureation

The behavior of the wake−up output pin is adjustable using the *Wake Pin Configuration* parameter. This parameter consists of various bitfields that configure how the wake−up indicates a match condition. These bitfields are as follows:

Bit 0	Active High / Active Low (0=Active High, 1=Active Low)
Bit 1	Momentary / Latched (0=Momentary, 1=Latched)
Bits 2:13	Momentary Timeout Value

When the wake−up pin is configured to be momentary, it transitions to the configured level (high or low depending on whether it is configured as active high or active low) and remains at that level for a fixed time frame before transitioning back to its normally−off state. The length of time the wake−up pin remains in the active state is a 12−bit

timeout value specified in bits 2 to 13 of the *Wake Pin Configuration* parameter (bits 14 and 15 are ignored). This value represents the number of timer increments the wake−up pin is active, where the timer base is 5 KHz. In other words, a timeout value of 1250 (the default) represents 1250/5000 seconds, or 250 ms. The maximum momentary timeout possible is just over 800 ms.

If configured as latched, it transitions to the configured level (high or low depending on whether it is configured as active high or active low) and remains at that level until it is manually reset by writing a 0 to the *Trigger State* parameter.

The current trigger state (matched or not matched) can be read at any time from the *Trigger State* parameter via I^2C.

Digital Microphone Configuration

The behavior of the digital microphone input is fixed and not configurable. When using the firmware built for a digital microphone, the algorithm always uses data from the left DMIC channel (`Left Data 0` in Figure 3), sampled on the rising edge of the DMIC_CLK signal, and no additional gain or attenuation is applied.

Table 18. ALGORITHM PARAMETERS

Parameter Number	Parameter Name	Read/Write	Parameter Description
0	Mode	Read/Write	The current mode (0=Sleep; 1=Standby, 2=Training, 3=Recognition)
1	Current Template	Read/Write	The currently−selected training template (0, 1, or 2)
2	Trigger State	Read/Write	Current trigger status (0=No Match, 1=Match)
3	Preamp Configuration	Read/Write	Analog microphone preamplifier gain setting (0=0dB, 1=3dB, 2=6dB, 3=9dB, 4=12dB, 5=15dB, 6=18dB, 7=21dB, 8=24dB, 9=27dB, 10=30dB)
4	VMIC Configuration	Read/Write	Microphone power (VMIC) output pin selection (0=GND, 1=1V, 2=2V, 3=HI−Z)
5	Wake Pin Configuration	Read/Write	Wake−up output pin configuration (bit[0]=active high (0)/low (1), bit[1]=momentary (0)/latched (1) bits[2:13]=momentary timeout in 0.2 ms increments)
15	Application Version	Read−only	The firmware version number (bits[15:12]=major, bits[11:8]=minor, bits[7:0]=revision)
16	Template Status	Read−only	The training template status word (bit[3]=Valid Distance, bit[2]=Template 2 Valid, bit[1]=Template 1 Valid, bit[0]=Template 0 Valid)
17	Template 0 Length	Read−only	The length (in frames) of training template 0
18	Template 1 Length	Read−only	The length (in frames) of training template 1
19	Template 2 Length	Read−only	The length (in frames) of training template 2
20	Intra−Template Distance 0	Read−only	The distance between template 0 and template 1
21	Intra−Template Distance 1	Read−only	The distance between template 0 and template 2
22	Intra−Template Distance 2	Read−only	The distance between template 1 and template 2
23	Intra−Template Distance Average	Read−only	The average intra−template distance
25	Decision Threshold	Read/Write	The match decision threshold
27	Distance to Template 0	Read−only	The distance between the last phrase and template 0
28	Distance to Template 1	Read−only	The distance between the last phrase and template 1
29	Distance to Template 2	Read−only	The distance between the last phrase and template 2
40	Frame Energy MSW	Read−only	The energy of the last frame (most significant word)
41	Frame Energy LSW	Read−only	The energy of the last frame (least significant word)
42	Short Term Energy MSW	Read−only	The average, short−term energy (most significant word)
43	Short Term Energy LSW	Read−only	The average, short−term energy (least significant word)
44	Noise Floor MSW	Read−only	The average noise floor (most significant word)
45	Noise Floor LSW	Read−only	The average noise floor (least significant word)

Company or Product Inquiries

For more information about ON Semiconductor products or services visit our Web site at http://onsemi.com .

Technical Contact Information

For technical support, email: dsp.support@onsemi.com

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81−3−5817−1050

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative