

AN987: BLUEGIGA I/O PROFILE

iWRAP application note

Thursday, 06 April 2017

Version 1.5



VERSION HISTORY

Version	Comment
1.0	First version
1.1	Fixed wrong packet length in "ADC set event interval" event
1.2	Fixed several response codes
1.3	Added information about unit of ADC event interval
1.4	Editorial changes
1.5	PIO read updated

TABLE OF CONTENTS

1	Introduction	4
2	iWRAP firmware overview	5
3	Using BGIO with iWRAP	7
3.1	Configuration	7
3.1.1	BGIO Sensor	7
3.1.2	BGIO Host	7
3.2	Service discovery	7
3.3	Connection establishment	8
3.4	Connection termination	8
3.5	General BGIO information	9
3.5.1	BGIO protocol	9
3.5.2	ADC read	10
3.5.3	ADC set event interval	11
3.5.4	PIO read	12
3.5.5	PIO write	13
3.5.6	PIO set event	14
3.5.7	PIO get direction	15
3.5.8	PIO set direction	16
3.5.9	PIO get bias	17
3.5.10	PIO set bias	18
3.5.11	Power saving	19

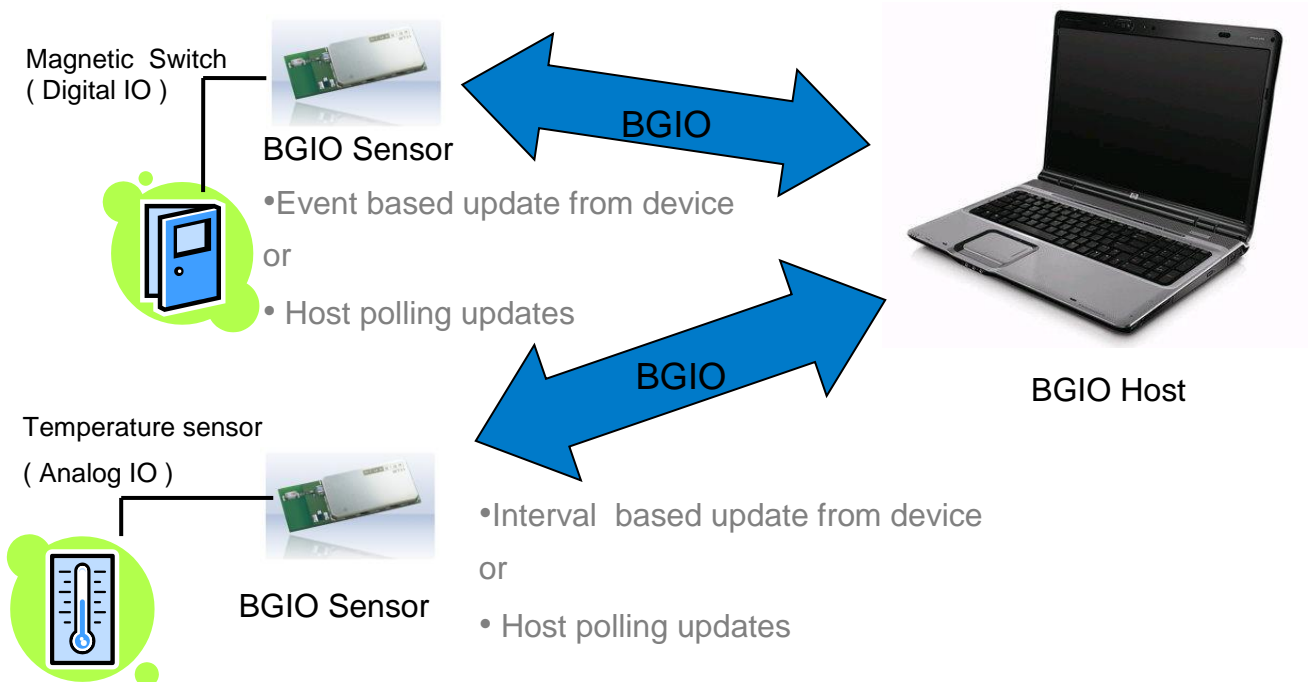
1 Introduction

This application note discusses Bluegiga I/O (BGIO) Profile's advantages and how the profiles can be utilized. Also practical examples are given how the BGIO can be used with the iWRAP firmware.

Bluegiga I/O profile is a proprietary profile which main purpose is to allow reading and setting of WT modules I/Os remotely over Bluetooth connection. This makes possible to build sensors which do not have host controller but instead only WT module and sensor component which can be for example a temperature meter.

BGIO defines two roles, that of a Sensor and Host:

- BGIO Sensor is the device who's GPIOs and/or AIOs are connected to sensors
- BGIO Host is the device which receives the measurements sent by the BGIO Sensor



2 iWRAP firmware overview

iWRAP is an embedded firmware running entirely on the RISC processor of WT12, WT12 and WT32 modules. It implements the full *Bluetooth* protocol stack and many *Bluetooth* profiles as well. All software layers, including application software, run on the internal RISC processor in a protected user software execution environment known as a Virtual Machine (VM).

The host system can interface to iWRAP firmware through one or more physical interfaces, which are also shown in the figure below. The most common interfacing is done through the UART interface by using the ASCII commands that iWRAP firmware supports. With these ASCII commands, the host can access *Bluetooth* functionality without paying any attention to the complexity, which lies in the *Bluetooth* protocol stack. GPIO interface can be used for event monitoring and command execution. PCM, SPDIF, I2S or analog interfaces are available for audio. The available interfaces depend on the used hardware.

The user can write application code to the host processor to control iWRAP firmware using ASCII commands or GPIO events. In this way, it is easy to develop *Bluetooth* enabled applications.

On WT32 there is an extra DSP processor available for data/audio processing.

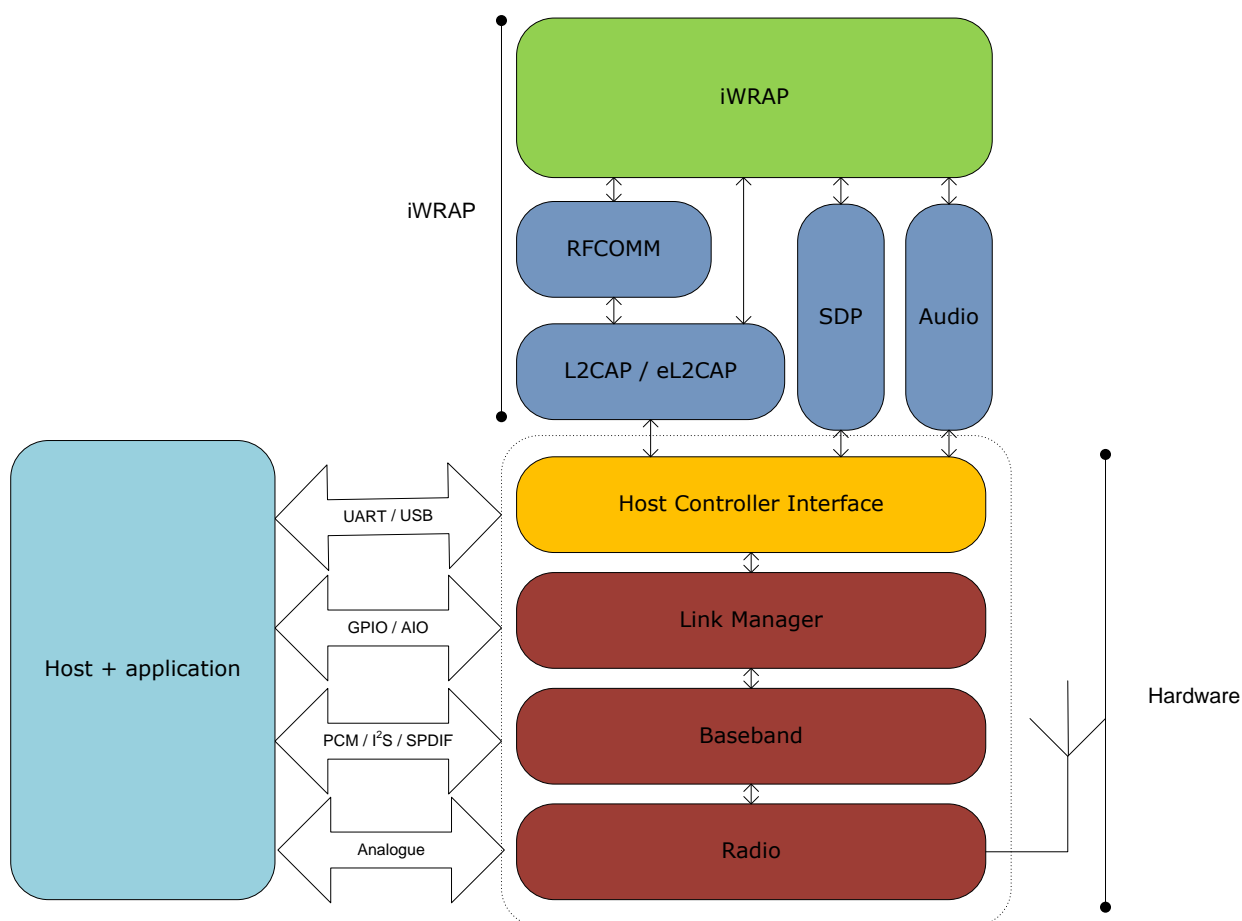


Figure 1: iWRAP Stack

In the figure above, a WRAP THOR *Bluetooth* module with iWRAP firmware could be connected to a host system for example through the UART interface. The options are:

- If the host system has a processor, software can be used to control iWRAP by using ASCII based commands or GPIO events.
- If there is no need to control iWRAP, or the host system does not need a processor, iWRAP can be configured to be totally transparent and autonomous, in which case it only accepts connections or automatically opens them.
- GPIO lines that WRAP THOR modules offer can also be used together with iWRAP to achieve additional functionality, such as Carrier Detect or DTR signaling.
- Audio interfaces can be used to transmit audio over a *Bluetooth* link.

3 Using BGIO with iWRAP

This chapter instructs the BGIO usage and configuration with the iWRAP firmware.

3.1 Configuration

3.1.1 BGIO Sensor

BGIO Sensor is enabled with command “**SET PROFILE BGIO {service_name}**”

service_name

This parameter configures user friendly description of the device. Neither special characters nor white spaces are allowed. Service name **ON** enables the profile with the default name.

A reset is needed to for the BGIO profile to become active.

Below is an example how to enable BGIO Sensor mode.

```
SET PROFILE BGIO ON
RESET
```

3.1.2 BGIO Host

BGIO Host doesn't require any pre-configurations to be made.

3.2 Service discovery

Bluetooth technology enables wireless service discovery, so you can find out the capabilities the remote device supports. Wireless service discovery uses the Bluetooth Service Discovery Profile (SDP).

With iWRAP the service discovery is performed with command: “**SDP {bd_addr} {uuid}**”.

bd_addr

Bluetooth device address of the remote device.

uuid

Universally unique identifier. Refers to the Bluetooth profile one wants to discover. For BGIO the **uuid** is af5c7d47-350b-45f6-bdf6-b403441edb77.

Below is an example how to perform a service discovery for BGIO Sensor device.

```
SDP 00:07:80:93:0c:aa af5c7d47-350b-45f6-bdf6-b403441edb77
SDP 00:07:80:93:0c:aa < I SERVICENAME S "Bluegiga IO" > < I PROTOCOLDESCRIPTORLIST < < U
L2CAP > < U RFCOMM I 05 > > >
SDP
```

Bluegiga IO

= Service name

05

= RFCOMM channel for BGIO

3.3 Connection establishment

To create BGIO connection to BGIO Sensor the BGIO Host needs to open connection using UUID af5c7d47-350b-45f6-bdf6-b403441edb77.

If the host device is iWRAP connection can be opened with **CALL** command:

"CALL {bd_addr} af5c7d47-350b-45f6-bdf6-b403441edb77 RFCOMM"

bd_addr Bluetooth device address of the device.

If connection establishment was successful you should receive CONNECT event:

"CONNECT {link_id} RFCOMM {channel} {bd_addr}"

link_id Local identifier for connection

channel Service channel on the remote device where connection was made

bd_addr Bluetooth device address of the device.

```
CALL 00:07:80:ff:ff:ff af5c7d47-350b-45f6-bdf6-b403441edb77 RFCOMM
CONNECT 0 RFCOMM 1 00:07:80:ff:ff:ff
```

3.4 Connection termination

Closing connection to the BGIO Sensor can be done by just closing the Bluetooth connection. If you are using iWRAP as the BGIO Host device you can do this with iWRAP command **"CLOSE {link_id}"**

link_id Numeric connection identified

BGIO connection termination.

```
CLOSE 0
NO CARRIER 0 ERROR 0
```


3.5 General BGIO information

This chapter contains general information and tips about the iWRAP and BGIO profile for the implementers.

3.5.1 BGIO protocol

BGIO uses a binary protocol that allows data transactions to be as short as possible, which allows more aggressive power saving methods to be used. There are three types of packets used in BGIO profile. Command packets are used by the Host to control the Sensor. Every command packet results a return packet sent by the Sensor back to the Host. Additionally, there are event packets which can be sent by the Sensor if it is configured to do so.

Byte	Function	Explanation
0	Packet Type	0x00 command 0x01 return 0x02 event
1	Packet Length	Combined length of "Command ID" and "Command Data" bytes
2	Command ID	0x00 ADC read 0x01 Reserved 0x02 ADC set event interval 0x03 PIO read 0x04 PIO write 0x05 PIO set event 0x06 PIO get direction 0x07 PIO set direction 0x08 PIO get bias 0x09 PIO set bias
3-	Command Data	Command specific data

Table 1: BGIO protocol syntax table

3.5.2 ADC read

Command **ADC read** gets readings from defined ADCs. Essentially does the same as **ADC set event interval**, but measurement events from each ADC come only once at the time the command is received by the BGIO Sensor.

Synopsis:

0x00 0x02 0x00 [source_mask]

Description:

source_mask	8bit Bit mask describing ADC from which a reading is required. For ADCs 1 and 2 you would use mask 0x03
--------------------	--

Response:

0x01 0x02 0x00 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters OK but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

Events:

{ 0x02 0x04 0x00 [source_id] [value] } { 0x02 0x04 0x00 [source_id] [value] } ...	Reading from 1st ADC defined by source_mask Reading from 2nd ADC defined by source_mask etc.
---	--

Description:

source_id	8bit Bit mask describing (single) ADC from which measurement was read
value	10bit reading (will be contained in 2 separate bytes)

3.5.3 ADC set event interval

Command **ADC set event interval** sets how often updated measurement information is sent by the Sensor to the Host from defined ADCs.

Synopsis:

0x00 0x06 0x02 [source_mask] [interval]

Description:

source_mask	8bit Bit mask describing ADC from which a reading is required
interval	32bit Value in milliseconds describing the interval between measurement updates

Response:

0x01 0x02 0x02 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

Events:

{ 0x02 0x04 0x02 [source_id] [value] }	Reading from 1st ADC defined by source_mask
{ 0x02 0x04 0x02 [source_id] [value] }	Reading from 2nd ADC defined by source_mask
...	etc.

Description:

source_id	8bit Bit mask describing (single) ADC from which measurement was read
value	10bit reading (will be contained in 2 separate bytes)

3.5.4 PIO read

Command **PIO read** gets status of defined remote digital GPIOs.

Synopsis:

0x00 0x03 0x03 [source_mask]

Description:

source_mask	16bit Bit mask describing PIOs from which a reading is required (obligatory parameter)
--------------------	---

Response:

0x01 0x04 0x03 [error_code] [value]	Return packet with PIO statuses from PIOs defined by source_mask
--	---

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
value	16bit reading

3.5.5 PIO write

Command **PIO write** sets status of defined remote digital GPIOs.

Synopsis:

0x00 0x05 0x04 [source_mask] [status]

Description:

source_mask	16bit Bit mask describing PIOs whose status needs to be changed
status	16bit new status (what is the example value for setting a GPIO high)

Response:

0x01 0x02 0x04 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

3.5.6 PIO set event

Command **ADC set event** sets which remote digital GPIO causes information about its status transition to be sent.

Synopsis:

0x00 0x03 0x05 [source_mask]

Description:

source_mask	16bit Bit mask describing PIOs which trigger event sending upon change of its state
--------------------	--

Response:

0x01 0x02 0x05 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

Events:

0x02 0x05 0x05 [pio_status] [pio_change]

Description:

pio_status	16bit current status of PIOs masked with source_mask
pio_change	16bit PIOs whose state change caused this event to be sent

3.5.7 PIO get direction

Command **PIO get direction** gets direction of defined PIOs.

Synopsis:

0x00 0x03 0x06 [source_mask]

Description:

source_mask	16bit Bit mask describing PIOs from which a reading is required
--------------------	--

Response:

0x01 0x04 0x06 [error_code] [value]	Return packet with PIO directions from PIOs defined by source_mask
--	---

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
value	16bit reading (1 = Output, 0 = Input)

3.5.8 PIO set direction

Command **PIO set direction** sets direction of defined PIOs.

Synopsis:

0x00 0x05 0x07 [source_mask] [direction]

Description:

source_mask	16bit Bit mask describing PIOs whose direction needs to be changed
direction	16bit new PIO directions (1 = Output, 0 = Input)

Response:

0x01 0x02 0x07 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

3.5.9 PIO get bias

Command **PIO get bias** gets bias of defined PIOs.

Synopsis:

0x00 0x03 0x08 [source_mask]

Description:

source_mask	16bit Bit mask describing PIOs from which bias reading is needed
--------------------	---

Response:

0x01 0x04 0x08 [error_code] [value]	Return packet with PIO directions from PIOs defined by source_mask
--	---

Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
value	16bit reading (1 = Strong pull up/down, 0 = weak pull up/down)

3.5.10 PIO set bias

Command **PIO set direction** sets direction of defined PIOs.

Synopsis:

0x00 0x05 0x09 [source_mask] [direction]

Description:

source_mask	16bit Bit mask describing PIOs whose bias needs to be changed
direction	16bit new PIO biases (1 = Output, 0 = Input)

Response:

0x01 0x02 0x09 [error_code]	Return packet (ACK)
--------------------------------------	---------------------

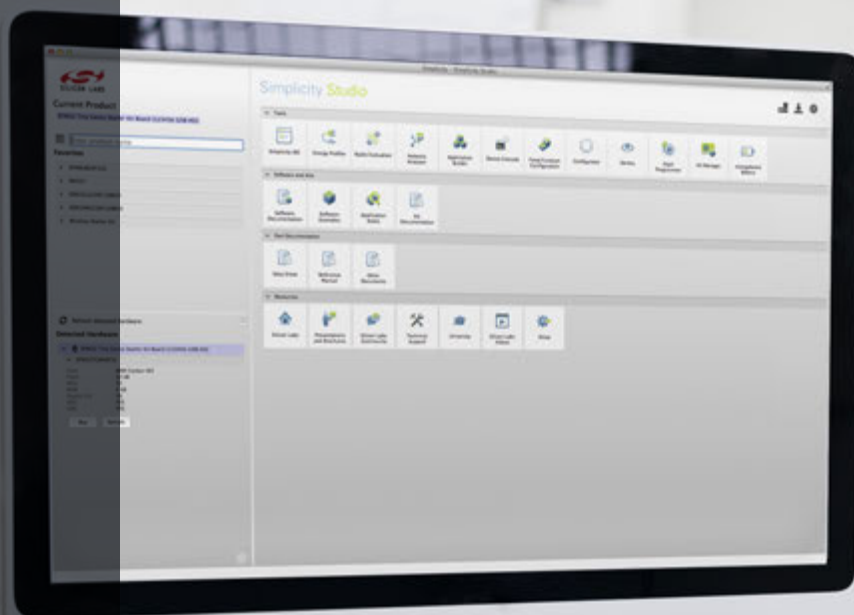
Description:

error_code	8bit 0x00 - OK, command completed successfully 0x01 - Fail, parameters ok but executing command failed 0x02 - Syntax error, parameter length is wrong 0x03 - Unknown command
-------------------	---

3.5.11 Power saving

iWRAP offers two power saving options. Sniff mode, which can be used to save power for active Bluetooth connections and deep sleep mode which puts the internal processor into a reduced duty cycle mode. Please refer to iWRAP user guide for more information about sniff and deep sleep modes.

One should also know that when Bluetooth connections are in active mode i.e. no power saving in use the master device uses 3-4 times less power than a slave device. Therefore for battery powered applications it might be useful to configure the device as a master rather than a slave. Look at following commands in iWRAP user guide: **SET {link_id} MASTER** or **SET {link_id} SLAVE** and **SET BT ROLE**.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



IoT Portfolio
www.silabs.com/loT



SW/HW
www.silabs.com/simplicity



Quality
www.silabs.com/quality



Support and Community
community.silabs.com

Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are not designed or authorized for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR®, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

<http://www.silabs.com>