UG439: Si3474 Software User's Guide

This software distribution contains demo code to interact with the Si3474 EVB. The demo code is supplied as Windows executables and requires no installation. Section 2. Hardware Configuration of this document describes the hardware setup and Section 3. Demos describes the demo code operation.

The demo code is written in Python and built upon a custom Python package which is included in the distribution. More advanced users can inspect the code, run the code from Python, or run their own Python scripts to interact with the Si3474. Section 4. Windows describes the installation process for Windows machines and Section 5. MacOS describes the process for MacOS.

RELATED DOCUMENTS

- UG425: Si3474 EVB User's Guide
- Si3474 Data Sheet

SUPPORTED OPERATING SYSTEMS

- Windows 7, 8, 10
- MacOS

Table of Contents

1. Package Contents

Unzip (extract) the distribution zip file to a folder to see the contents.

1.1 Content Tree

- **demo**
  - auto_mode_4p.exe
  - auto_mode_2p.exe
  - semi_auto_mode_4p.exe
  - semi_auto_mode_2p.exe
- **develop**
  - **doc**
  - **examples**
  - **silabs_pse**
  - install.bat
  - silabs_pse-x.y.z-py3-none-any.whl
  - silabs_pse-x.y.z.tar.gz
- **firmware**
  - update_firmware.exe
  - update_firmware.py
  - si3474a_factory_combo_bl_firmware_image.c
- LICENCE.txt
- README.txt

1.2 Content Description

Content	Description
demo	Contains Windows executable files (.exe) which can be run without any Python installation. See Section 3. Demos for descriptions of each demo.
develop	Contains Python example source code, silabs_pse Python package, and software documentation.
doc	Contains documentation on the silabs_pse Python package. To view, open the index.html file in a web browser (e.g., Chrome).
examples	Python example source code, which has been compiled into the demo executable files. See Section 3. Demos for descriptions of each demo.
install.bat	(Windows only) This batch script will install the silabs_pse Python package into a Python interpreter using pip (https://packaging.python.org/key_projects/#pip).
silabs_pse	This is the source code of the silabs_pse package which can be installed into a Python interpreter.
silabs_pse-x.y.z-py3-none-any.whl	This "wheel" file contains the silabs_pse package in binary format which can be installed into a Python interpreter using pip (https://packaging.python.org/key_projects/#pip). See Section 4.2.2 Install silabs_pse Package for installing with Windows. See Section 5.1.2 Install silabs_pse Package for installing with MacOS.

2. Hardware Configuration

To run the demos, connect the EVB in the configuration shown in Figure 2.1 Hardware Configuration for Demo (4P) on page 4. Use the four-pair powered configuration shown in Figure 2.2 JP1-JP8 Configuration for Four-Pair Powered Operation on page 5 for demos denoted with 4P. Use the two-pair powered configuration shown in Figure 2.4 JP1-JP8 Configuration for Two-Pair Powered Operation on page 6 for the demos denoted with 2P.

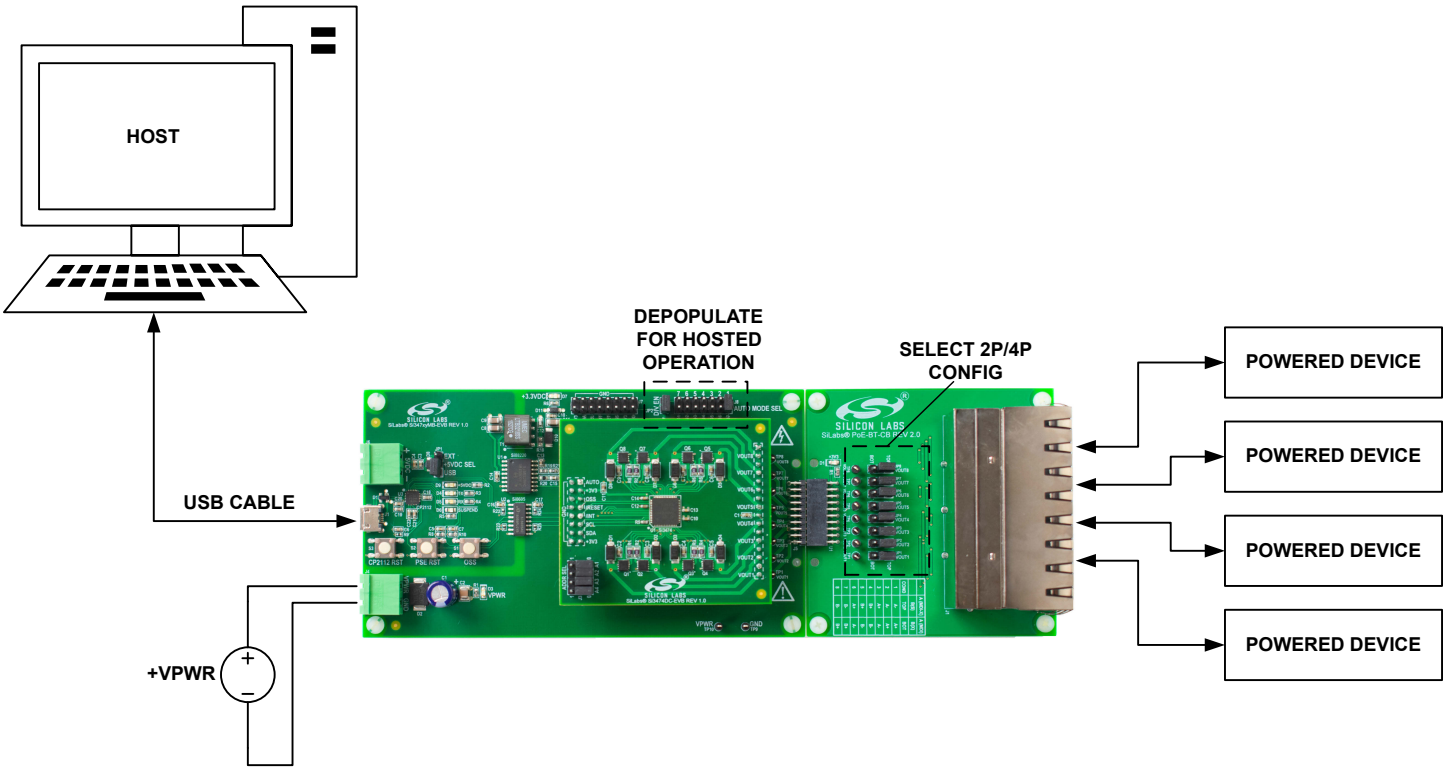


Figure 2.1. Hardware Configuration for Demo (4P)

2.1 Configuring for 4P Operations

To configure the EVB to power over four-pairs (802.3bt), configure the jumpers on the PoE-BT-CB as shown in Figure 2.2 JP1-JP8 Configuration for Four-Pair Powered Operation on page 5. This configuration should be used for all demos noted as 4P.

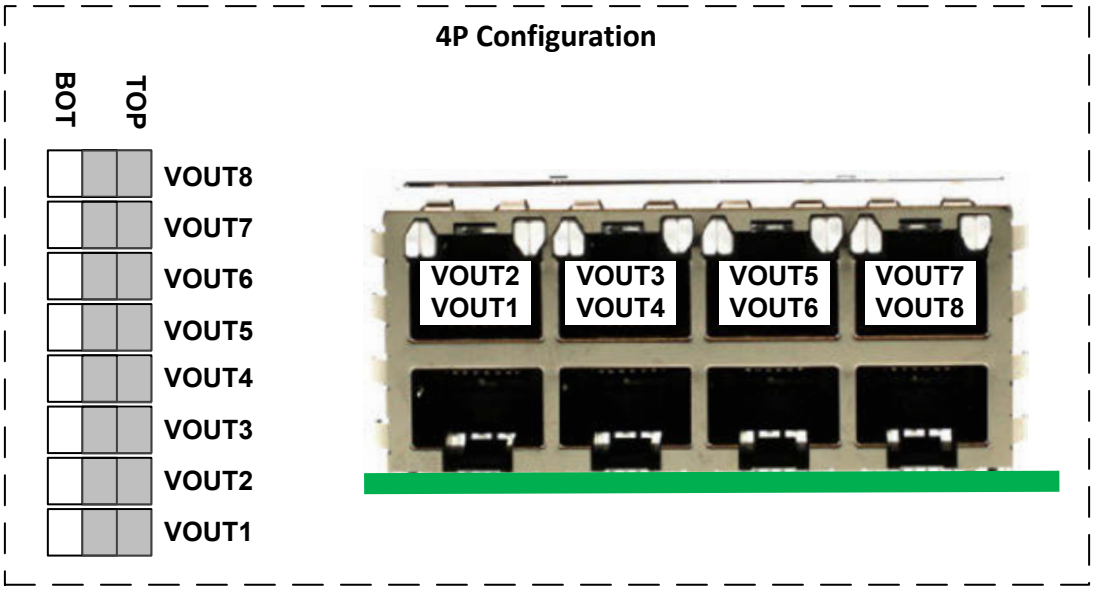


Figure 2.2. JP1-JP8 Configuration for Four-Pair Powered Operation

Note that in the software distribution and demos, all ports and port pairs are indexed from zero. In 4P configuration, ports and port pairs are numbered as shown in Figure 2.3 Mapping Between Hardware Indexing (from one) and Software Indexing (from zero) in 4P Configuration on page 5 below. Each port pair is made up of two ports.

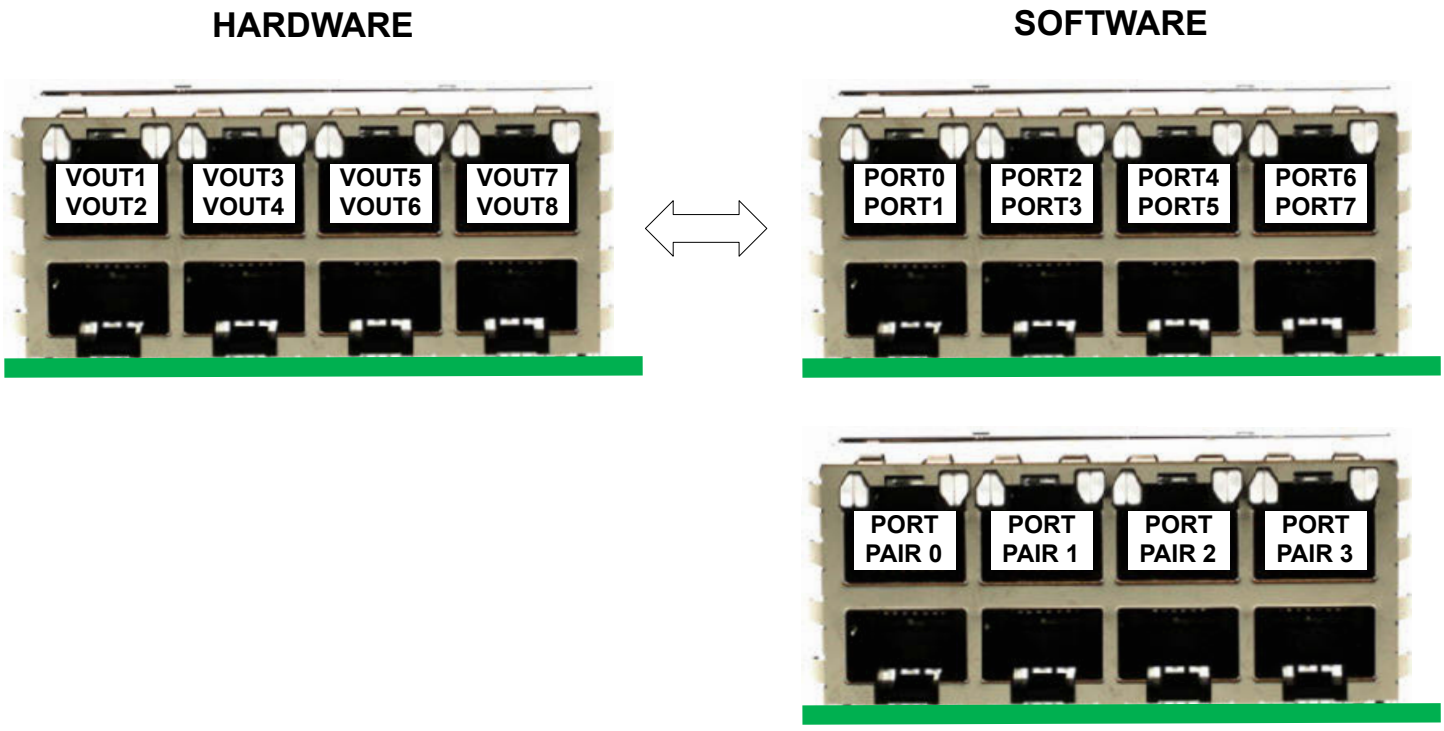


Figure 2.3. Mapping Between Hardware Indexing (from one) and Software Indexing (from zero) in 4P Configuration

2.2 Configuring for 2P Operations

To configure the EVB to power over two-pairs (802.3at), configure the jumpers on the PoE-BT-CB as shown in Figure 2.4 JP1-JP8 Configuration for Two-Pair Powered Operation on page 6. This configuration should be used for all demos noted as 2P.

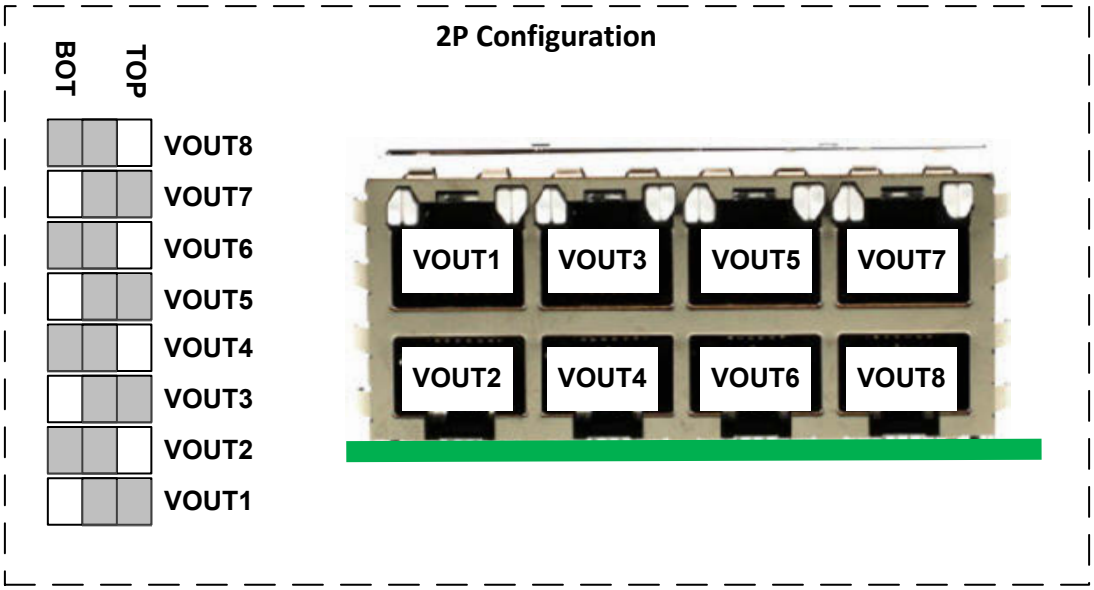


Figure 2.4. JP1-JP8 Configuration for Two-Pair Powered Operation

Note that in the software distribution and demos, all ports and port pairs are indexed from zero. In 2P configuration, ports are numbered as shown in Figure 2.5 Mapping Between Hardware Indexing (from one) and Software Indexing (from zero) in 2P Configuration on page 6

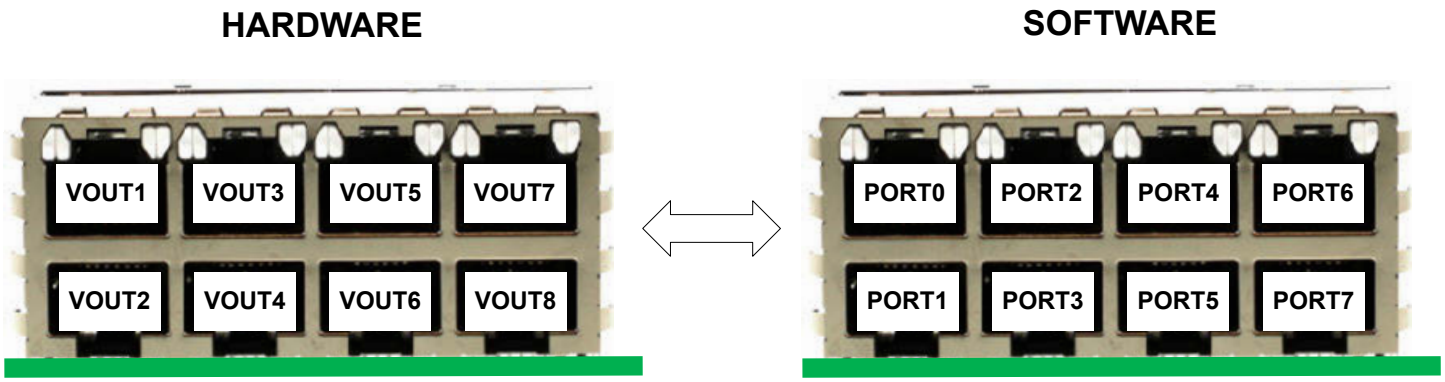


Figure 2.5. Mapping Between Hardware Indexing (from one) and Software Indexing (from zero) in 2P Configuration

3. Demos

Demo source code is located in **develop/examples**.

3.1 Auto Mode (4P)

This demo configures the PSE into its fully autonomous mode which allows it to detect, classify, and power PDs without any host control. The PSE is configured to provide power over four pairs with 90W allocation per port pair. The screen will continuously display the detection, classification, and power consumption information for all ports.

```
PSE Temperature: 31.51 C
VPWR Voltage: 55.9 V
Port 0 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_7_4P_SS     Assigned: CLASS_7_4P_SS     Power: 0.58W
Port 1 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_7_4P_SS     Assigned: CLASS_7_4P_SS     Power: 0.57W
Port 2 Detect: OPEN_CIRCUIT
Port 3 Detect: OPEN_CIRCUIT
Port 4 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_8_4P_SS     Assigned: CLASS_8_4P_SS     Power: 4.32W
Port 5 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_8_4P_SS     Assigned: CLASS_8_4P_SS     Power: 5.17W
Port 6 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_4           Assigned: CLASS_4           Power: 0.23W
Port 7 Detect: VALID CC_SINGLE_SIG     Requested: CLASS_4           Assigned: CLASS_4           Power: 0.19W
```

In this example, there are 3 different four-pair powered devices connected to the EVB. The detection, requested class, assigned class, and power consumption of each PD is displayed and updated in real time. The port pair with no PD attached displays the detection status of OPEN_CIRCUIT for both ports, indicating that no load has been detected. Detection is continuously performed on these ports, and the screen will be updated upon detecting a PD.

3.2 Auto Mode (2P)

Note: The PoE-BT-CB jumper settings must be configured to support 2-pair powering for this demo to work as intended. See Section 2.2 Configuring for 2P Operations for configuring the jumper settings to 2-pair powered ports.

This demo is identical to the Section 3.1 Auto Mode (4P) demo, except the PSE is configured to provide power over two pairs with 30W power allocation per port. This allows powering up to 8 different PDs (each up to 30W) at once.

```
PSE Temperature: 30.86 C
VPWR Voltage: 55.9 V
Port 0 Detect: VALID                   Requested: CLASS_2           Assigned: CLASS_2           Power: 1.14W
Port 1 Detect: VALID                   Requested: CLASS_3           Assigned: CLASS_3           Power: 6.30W
Port 2 Detect: OPEN_CIRCUIT
Port 3 Detect: VALID                   Requested: CLASS_3           Assigned: CLASS_3           Power: 6.38W
Port 4 Detect: OPEN_CIRCUIT
Port 5 Detect: OPEN_CIRCUIT
Port 6 Detect: VALID                   Requested: CLASS_4           Assigned: CLASS_4           Power: 0.43W
Port 7 Detect: OPEN_CIRCUIT
```

In this example, there are 4 different two-pair powered devices connected to the EVB. The detection, requested class, assigned class, and power consumption of each PD is displayed and updated in real time. The ports with no PD attached display the detection status of OPEN_CIRCUIT, indicating that no load has been detected. Detection is continuously performed on these ports, and the screen will be updated upon detecting a PD.

3.3 Semi-Auto Mode (4P)

This demo configures the PSE into its semi-autonomous mode which allows it to detect and classify PDs without host control, and relies on a host to power PDs. The demo code functions as the host, and will close the loop with the PSE to react to events and send push-button commands for powering PDs. The screen will display all events received by the host from the PSE and actions taken by the host.

```
I2C 0x20 REG POWER_ALLOCATION 0x29 = 0xff
I2C 0x21 REG POWER_ALLOCATION 0x29 = 0xff
I2C 0x20 REG PORT_REMAP 0x26 = 0xe4
I2C 0x21 REG PORT_REMAP 0x26 = 0xe4
I2C 0x20 REG MISC 0x17 = 0x8c
I2C 0x21 REG MISC 0x17 = 0x8c
I2C 0x20 REG PORT_MODE 0x12 = 0xaa
I2C 0x21 REG PORT_MODE 0x12 = 0xaa
I2C 0x20 REG DETECT_CLASS_ENABLE 0x14 = 0xff
I2C 0x21 REG DETECT_CLASS_ENABLE 0x14 = 0xff
I2C 0x20 REG INTERRUPT 0x00 = 0x80
I2C 0x20 REG SUPPLY_EVENT_COR 0x0b = 0x30
I2C 0x21 REG INTERRUPT 0x00 = 0x80
I2C 0x21 REG SUPPLY_EVENT_COR 0x0b = 0x30
I2C 0x20 REG INTERRUPT 0x00 = 0x08
I2C 0x20 PORT 0 DETECT DONE: STATUS: VALID - CC_SINGLE_SIG
I2C 0x20 PORT 1 DETECT DONE: STATUS: VALID - CC_SINGLE_SIG
I2C 0x20 REG INTERRUPT 0x00 = 0x10
I2C 0x20 PORT 0 CLASS DONE: STATUS: CLASS_8_4P_SS
I2C 0x20 SENDING PUSH-BUTTON ON TO PORT PAIR 0
I2C 0x20 PORT 1 CLASS DONE: STATUS: CLASS_8_4P_SS
I2C 0x20 REG INTERRUPT 0x00 = 0x08
I2C 0x20 PORT 0 DETECT DONE: STATUS: VALID - CC_SINGLE_SIG
I2C 0x20 PORT 1 DETECT DONE: STATUS: VALID - CC_SINGLE_SIG
I2C 0x20 REG INTERRUPT 0x00 = 0x10
I2C 0x20 PORT 0 CLASS DONE: STATUS: CLASS_8_4P_SS
I2C 0x20 PORT 1 CLASS DONE: STATUS: CLASS_8_4P_SS
I2C 0x20 REG INTERRUPT 0x00 = 0x01
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x03
I2C 0x20 REG INTERRUPT 0x00 = 0x02
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x30
I2C 0x20 REG INTERRUPT 0x00 = 0x07
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x33
I2C 0x20 REG DISCONNECT_PCUT_FAULT_COR 0x07 = 0x30
```

The semi-auto demo always begins with initialization. The PSE is first configured to four-pair powering with a power allocation of 90W per port pair, as shown in the POWER_ALLOCATION register. Next, the port remap is written to its default value, which is a one-to-one map between logical and physical ports. The MISC register is configured to enable the interrupt pin, detect change, and class change bits. All ports are configured into semi-autonomous mode, as shown in the PORT_MODE register. Finally, detection and classification are enabled for all ports, as shown in the DETECT_CLASS_ENABLE register.

After initialization, interrupts and events are handled. The PSE always has a UVLO event upon boot-up, as shown in the INTERRUPT and SUPPLY_EVENT registers. After reading the supply event clear-on-read (COR) register, these events are cleared and no further SUPPLY_EVENT interrupts are received.

In this example, the user plugs in a PD to the EVB. After plugging in the PD, the first interrupt received and printed to screen is a DETECT_CC_DONE interrupt. The interrupt is handled by reading the CLASS_DETECT_EVENT register, which indicates that detection and connection check have been completed on ports 0 and 1. Next, the CLASS_DETECT_STATUS and AUTOCLASS_CONNECTION_CHECK registers are read for the associated ports, and the results are printed to the screen. In this example, it is shown that a load was detected with a valid detection status and connection check result of single signature for both ports.

The next interrupt received and printed to screen is a CLASS_DONE interrupt. The interrupt is handled by reading the CLASS_DETECT_EVENT register, which indicates that classification has been completed on port 0. Next, the CLASS_DETECT_STATUS register is read for the associated port, and the results are printed to the screen. In this example, it is shown that the connected load was classified as requesting class 8 power.

Upon receiving valid detection and classification events, the demo code powers the PD by writing to the push-button register PB_POWER_ENABLE for the associated port pair. The PSE receives the push-button power on command and performs detection and classification again before applying power to the PD. As shown in the print statements, the detection and classification interrupts and results are identical to before. After classification, the next interrupt received is a POWER_ENABLE_CHANGE event. This event is handled by reading the POWER_EVENT register, which shows that power has been enabled for ports 0 and 1. Power enabled indicates that the ports have been turned on and the PD is in the inrush phase of power-up. The next interrupt received is a POWER_GOOD_CHANGE

event. This event is handled by reading the POWER_EVENT register, which shows that the power status of ports 0 and 1 is now POWER_GOOD. This indicates that the PD has exited inrush and power has been successfully applied to the PD.

Finally, the user unplugs the PD from the EVB. The interrupts DISCONNECT, POWER_ENABLE_CHANGE, and POWER_GOOD_CHANGE are received simultaneously. The POWER_ENABLE_CHANGE and POWER_GOOD_CHANGE interrupts are handled by reading the POWER_EVENT register, which shows that power status has changed for ports 0 and 1. The DISCONNECT interrupt is handled by reading the DISCONNECT_PCUT_FAULT register, which shows that a disconnection event has occurred for ports 0 and 1.

The PSE has successfully powered a PD and then removed power upon detecting that the PD has been disconnected.

3.4 Semi-Auto Mode (2P)

Note: The PoE-BT-CB jumper settings must be configured to support 2-pair powering for this demo to work as intended. See Section 2.2 Configuring for 2P Operations for configuring the jumper settings to 2-pair powered ports.

This demo is identical to the Section 3.3 Semi-Auto Mode (4P) demo, except the PSE is configured to provide power over two pairs with 30W power allocation per port. This allows powering up to 8 different PDs (each up to 30W) at once.

```
I2C 0x20 REG POWER_ALLOCATION 0x29 = 0x33
I2C 0x21 REG POWER_ALLOCATION 0x29 = 0x33
I2C 0x20 REG PORT_REMAP 0x26 = 0xe4
I2C 0x21 REG PORT_REMAP 0x26 = 0xe4
I2C 0x20 REG MISC 0x17 = 0x8c
I2C 0x21 REG MISC 0x17 = 0x8c
I2C 0x20 REG PORT_MODE 0x12 = 0xaa
I2C 0x21 REG PORT_MODE 0x12 = 0xaa
I2C 0x20 REG DETECT_CLASS_ENABLE 0x14 = 0xff
I2C 0x21 REG DETECT_CLASS_ENABLE 0x14 = 0xff
I2C 0x20 REG SUPPLY_EVENT_COR 0x0b = 0x30
I2C 0x21 REG SUPPLY_EVENT_COR 0x0b = 0x30
I2C 0x20 PORT 0 DETECT DONE: STATUS: VALID
I2C 0x20 PORT 0 CLASS DONE - STATUS: CLASS_4
I2C 0x20 SENDING PUSH-BUTTON ON TO PORT 0
I2C 0x20 PORT 0 DETECT DONE: STATUS: VALID
I2C 0x20 PORT 0 CLASS DONE - STATUS: CLASS_4
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x01
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x10
I2C 0x20 REG POWER_EVENT_COR 0x03 = 0x11
I2C 0x20 REG DISCONNECT_PCUT_FAULT_COR 0x07 = 0x10
```

As with the 4P demo, the semi-auto demo always begins with initialization. The PSE is first configured to two-pair powering with a power allocation of 30W per port, as shown in the POWER_ALLOCATION register. Next, the port remap is written to its default value, which is a one-to-one map between logical and physical ports. The MISC register is configured to enable the interrupt pin, detect change, and class change bits. All ports are configured into semi-autonomous mode, as shown in the PORT_MODE register. Finally, detection and classification are enabled for all ports, as shown in the DETECT_CLASS_ENABLE register.

After initialization, interrupts and events are handled. The PSE always has a UVLO event upon boot-up, as shown in the SUPPLY_EVENT register. After reading the supply event clear-on-read (COR) register, these events are cleared and no further SUPPLY_EVENT interrupts are received.

In this example, the user plugs in a PD to the EVB. After plugging in the PD, the first interrupt received is a DETECT_CC_DONE interrupt. The interrupt is handled by reading the CLASS_DETECT_EVENT register, which indicates that detection has been completed on port 0. Next, the CLASS_DETECT_STATUS register is read for the associated port, and the results are printed to the screen. In this example, it is shown that a load was detected with a valid detection status on port 0. Note that because the port is configured for two-pair powering, no connection check is performed.

The next interrupt received is a CLASS_DONE interrupt. The interrupt is handled by reading the CLASS_DETECT_EVENT register, which indicates that classification has been completed on port 0. Next, the CLASS_DETECT_STATUS register is read for the associated port, and the results are printed to the screen. In this example, it is shown that the connected load was classified as requesting class 4 power.

Upon receiving valid detection and classification events, the demo code powers the PD by writing to the push-button register PB_POWER_ENABLE for the associated port. The PSE receives the push-button power on command and performs detection and classification again before applying power to the PD. As shown in the print statements, the detection and classification results are identical to before. After classification, the next interrupt received is a POWER_ENABLE_CHANGE event. This event is handled by reading the POWER_EVENT register, which shows that power has been enabled for port 0. Power enabled indicates that the port has been turned on and the PD is in the inrush phase of power-up. The next interrupt received is a POWER_GOOD_CHANGE event. This event is handled by reading the POWER_EVENT register, which shows that the power status of port 0 is now POWER_GOOD. This indicates that the PD has exited inrush and power has been successfully applied to the PD.

Finally, the user unplugs the PD from the EVB. The interrupts DISCONNECT, POWER_ENABLE_CHANGE, and POWER_GOOD_CHANGE are received simultaneously. The POWER_ENABLE_CHANGE and POWER_GOOD_CHANGE interrupts are handled by reading the POWER_EVENT register, which shows that power status has changed for ports 0 and 1. The DISCONNECT interrupt is handled by reading the DISCONNECT_PCUT_FAULT register, which shows that a disconnection event has occurred for port 0.

The PSE has successfully powered a PD and then removed power upon detecting that the PD has been disconnected.

4. Windows

4.1 Run Demo Scripts

No installation is necessary to run the demos on Windows. Navigate to the **demo** folder and run the .exe files directly. The source code of the demos can be found in the **develop/examples** directory. To run manually, see Section 4.3 Run Demo Scripts Manually.

4.2 Installation

Note: No installation is required to run the demos on Windows. See Section 4.1 Run Demo Scripts.

4.2.1 Install Python

Python must already be installed to use the silabs_pse package and run the example scripts using the Python interpreter. Download and install Python 3 from the Python website: https://www.python.org/downloads/

4.2.2 Install silabs_pse Package

The **install.bat** file in the **develop** directory will complete this step automatically. To run, simply double-click it.

Otherwise, use pip to install the silabs_pse Python package into a Python interpreter using the .whl file included in the distribution. Navigate to the **develop** directory in the distribution in a shell or CMD window and execute the following command (replace x.y.z with the appropriate version included in the distribution):

```
pip install silabs_pse-x.y.z-py3-none-any.whl
```

4.3 Run Demo Scripts Manually

First, make sure Python and the silabs_pse package are installed following the Section 4.2 Installation steps. The demo scripts can be run directly using the Python Interpreter. Navigate to the **develop** directory in the distribution in a shell or CMD window and execute the following command, replacing the script with the desired demo script.

```
python auto_mode_4p.py
```

The script should run and display in the console window.

Note: Integrated Development Environments (IDE) can be useful for developing and running code in a single window. PyCharm is a popular choice for Python development

4.4 Write Custom Scripts

Custom scripts can be written and executed directly using the Python interpreter like the demo scripts. See the silabs_pse API documentation in **develop/doc** for programmer resources.

5. MacOS

5.1 Installation

5.1.1 Install Python

Python must already be installed to use the silabs_pse package and run the example scripts. Download and install Python 3 from the Python website: https://www.python.org/downloads/

5.1.2 Install silabs_pse Package

Use pip to install the Python package into a Python interpreter using the .whl file included in the distribution. In terminal, navigate to the **develop** directory of the distribution and execute the following command (replace x.y.z with the appropriate version included in the distribution):

```
pip install silabs_pse-x.y.z-py3-none-any.whl
```

5.2 Run Demo Scripts

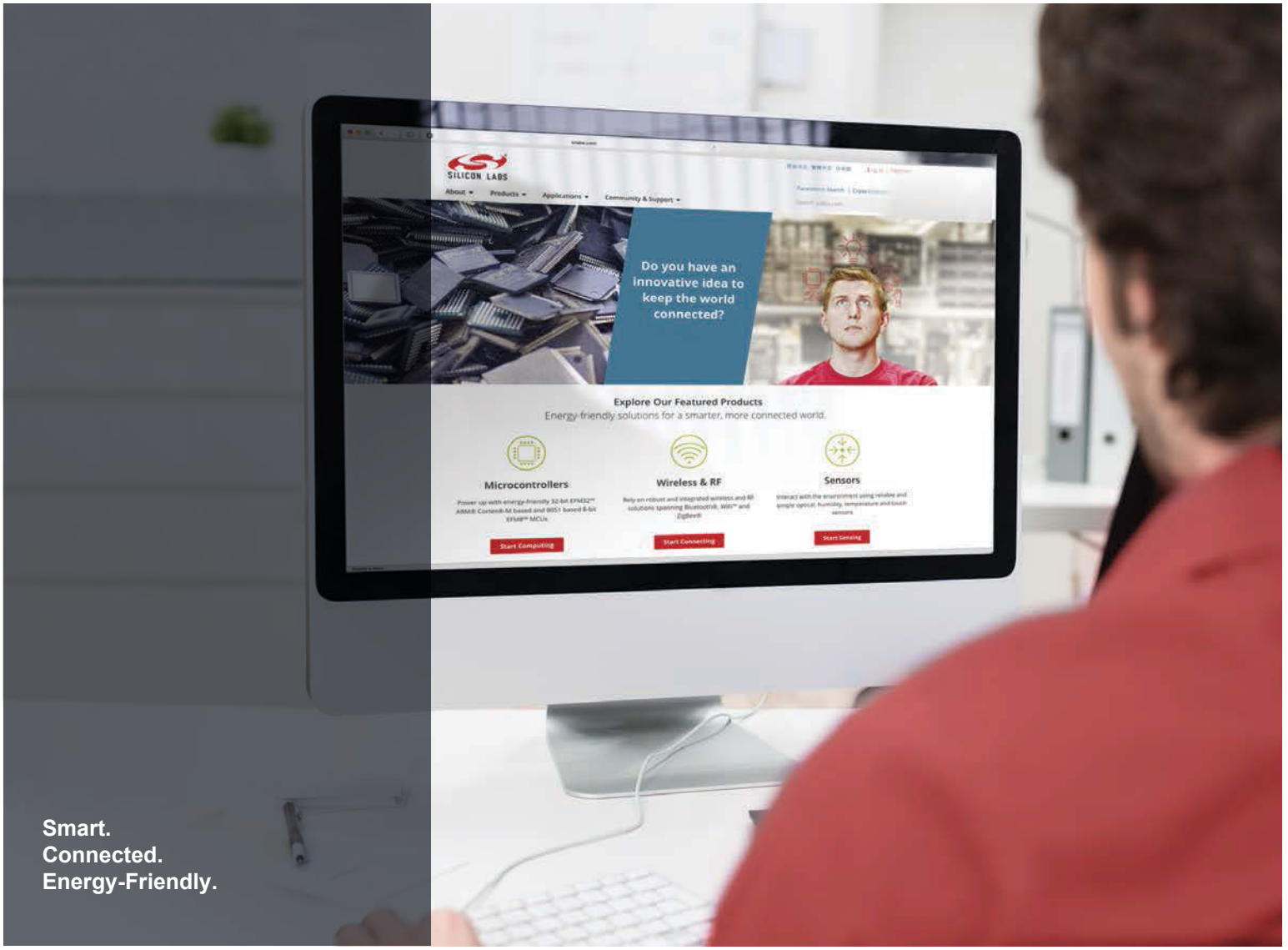
After installation, in terminal navigate to the **develop/examples** directory and run the following command, pointing to the desired example script:

```
python auto_mode_4p.py
```

The script should run and display in the console window.

5.3 Write Custom Scripts

Custom scripts can be written and executed in Python like the demo scripts. See the silabs_pse API documentation in **develop/doc** for programmer resources.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com