Porting Guide

Yanzhao Zhang edited this page on Nov 30, 2022 · 85 revisions

Description/Scope

SONiC is designed to be portable to a variety of network devices. Among devices, the majority of platform-dependent code involves controlling the ASIC, which SONiC handles via the Switch Abstraction Interface (SAI). However, many devices share the same ASIC platform, and only differ in other device-specific hardware components which require custom device drivers and/or configuration files which are loaded when the system is initialized. This guide describes requirements and general guidelines to follow when porting SONiC to a new device *which contains a supported ASIC platform*. Therefore, the remainder of this document assumes the device you are porting for contains an ASIC platform which is already supported in SAI/SONiC.

Introduction

As mentioned above, to port SONiC to a new device, you will need to provide device-specific hardware drivers as well as device-specific configuration files to initialize your device properly. All device-specific changes will be made in the sonic-buildimage repository.

Platform Drivers

You are required to provide drivers to expose your device-specific hardware via sysfs to allow SONiC to communicate with them. Below is a list of drivers that are required along with their required feature sets.

- QSFP transceivers
    - Read from/write to transceiver EEPROM
        - **NOTE: SONiC currently contains the `optoe` driver. It is highly recommended to use this existing driver to expose transceiver EEPROMs via sysfs in a uniform manner**
    - Enable/disable low-power mode
    - Reset transceiver
    - Query transceiver module presence
    - Detect interrupt upon transceiver plug and unplug events
- Sensors
    - Query temperatures, fan speeds, voltages, etc. (for generating alarms at critical thresholds)
- Front panel port status LEDs

    - Control LED state

These modules should be placed in the appropriate directory under the `platform/` directory here.

SONiC images are compiled for each ASIC vendor, and as such are designed to be installed on any supported device running that vendor's ASIC. For example, a sonic-broadcom.bin image is designed to be installed on all supported devices which implement a Broadcom ASIC. Therefore, all platform modules for all devices which implement that vendor's ASIC are compiled into one image. The appropriate platform drivers are installed on the first boot of the device upon detection of the device's SKU.

In your module's .mk makefile you are able to specify which Debian package(s) are built and compiled into the images for which platform (by ONIE platform string). Also note that the same platform driver can be installed on multiple platforms; simply specify each platform in your module's .mk makefile.

Device-specific platform drivers added to SONiC must abide by the following rules:

1. Drivers must be packaged into one or more Debian packages (.deb)
2. All dependencies of the module must be specified in the Debian package, as they will be downloaded and compiled into the SONiC image, *not* at install time
3. Drivers must **not** require a reboot after installation
4. Drivers must support both initialization **and** deinitialization (required to allow for updating drivers on a running system in the future)

SONiC Platform API Package

SONiC needs a way to interface with the unique configuration of peripheral hardware present in each platform (fans, LEDs, power supply units, SFP transceivers, thermal sensors, etc.). This is handled by the SONiC Platform API. SONiC contains a number of daemons and other applications which collect data from the peripheral devices and also write to the devices (e.g., read temperature data from thermal sensors and write new fan speeds to fans). This interaction is abstracted by the SONiC Platform API.

The SONiC Platform API is an object-oriented API which is designed to reflect the physical composition of a network device. At the root level of the API is the Platform object. The Platform contains a Chassis object. The Chassis object, in turn, can contain an array of other hardware devices (fans, LEDs, power supply units, SFP transceivers, thermal sensors, etc.). If the platform's chassis is modular, the Chassis can also contain objects which represent the various modules (line cards, fabric cards, etc.).

When porting SONiC to a new platform, one must create a concrete implementation of a `sonic-platform` Python 3 wheel package. The classes in this package must inherit from

the abstract base classes defined in the sonic-platform-base Python package. The sonic-platform Python package is expected to be generated at the time of building the platform device drivers. The package file should be named sonic_platform-1.0-py3-none-any.whl and should contain a file hierarchy similar to the following (note that additional files may be added as necessary to hold any shared code and avoid code duplication):

```
sonic_platform/
|-- __init__.py
|-- chassis.py
|-- component.py
|-- eeprom.py
|-- fan.py
|-- fan_drawer.py
|-- led.py
|-- platform.py
|-- psu.py
|-- sfp.py
|-- thermal.py
|-- watchdog.py
```

After the sonic-platform package is built, it should be installed in the host OS by calling pip3 install sonic_platform-1.0-py3-none-any.whl, and then the sonic_platform-1.0-py3-none-any.whl file should be copied to the appropriate device/<VENDOR_NAME>/<ONIE_PLATFORM_STRING>/ directory. When the PMon (platform monitor) container starts up, it will look for the wheel file in that location and install it if it is not yet installed.

The SONiC Platform API may need root permission to access certain platform sysfs entries when it is called in the host OS, but it should be executed with any permission issue in PMon container.

Testing/Validation

NOTE: This section is a work-in-progress

- Ensure all platform daemons function properly
    - TODO
- Ensure the sfputil utility can properly communicate with transceivers
    - Call the sfputil application and confirm the following functionality is as expected
        - sfputil show eeprom / sfputil show eeprom --dom: Verify all transceivers' EEPROM data is displayed properly

        - `sfputil show presence`: Verify all transceivers' presence status is displayed properly
        - `sfputil show lpmode`: Verify all transceivers' low-power mode status is displayed properly
        - `sfputil reset ...`: Attempt to reset every transceiver, verify all transceivers reset properly
        - `sfputil lpmode ...`: Enable low-power mode on all transceivers, verify all transceivers have low-power mode enabled. Disable low-power mode on all transceivers, verify all transceivers have low-power mode disabled.

Device-Specific Files

All files that contain configuration or diagnostic information that are specific to a particular platform or hardware SKU reside under the `device/` subdirectory here. The hierarchy of which is described in the following section.

Device-Specific File Directory Structure

SONiC distinguishes devices by a hierarchy of two levels, *Platform* and *Hardware SKU*. Some devices share the majority of the same hardware and only differ in, for example, the number/type of front panel ports (due to breakout configurations, etc.). In other words, multiple *hardware SKUs* can be configured on top of the same *platform*. While the front panel port configurations of these devices may differ, they still share common platform hardware like fans, sensors and system EEPROMs. The SONiC directory structure for device-specific files reflects this hierarchy and is designed to reduce the need for duplicate files as follows:

- The `device/` directory contains one subdirectory per device vendor, named `<VENDOR_NAME>/` (e.g., `dell`, `mellanox`).
    - Each `<VENDOR_NAME>/` directory contains one subdirectory per unique ONIE platform string, named `<ONIE_PLATFORM_STRING>/` (e.g., `x86_64-dell_s6000_s1220-r0`, `x86_64-mlnx_msn2700-r0`) and contains configuration files shared by *all* hardware SKUs built upon that platform.
        - Each `<ONIE_PLATFORM_STRING>/` directory contains one subdirectory per unique hardware SKU, named `<HARDWARE_SKU>/` (e.g., `Force10-S6000`, `ACS-MSN2700`) and contains configuration files specific to that individual hardware SKU.

```
device/
|-- <VENDOR_NAME>/
```

```
|   |-- <ONIE_PLATFORM_STRING>/
|   |   |-- <HARDWARE_SKU>/
|   |   |   |-- port_config.ini [DEPRECATED if platform.json/hwsku.json are in place]
|   |   |   |-- hwsku.json
|   |   |   |-- sai.profile
|   |   |   |-- xxx.config.bcm
|   |   |   |-- buffer_defaults_t0/t1.j2
|   |   |   |-- pg_profile_lookup.ini
|   |   |   |-- Qos.json
|   |   |-- plugins/ [DEPRECATED]
|   |   |   |-- led_control.py
|   |   |-- default_sku
|   |   |-- fancontrol [DEPRECATED in favor of utilizing thermalctld]
|   |   |-- installer.conf
|   |   |-- led_proc_init.soc
|   |   |-- pcie.yaml
|   |   |-- platform_asic
|   |   |-- platform_env.conf
|   |   |-- platform.json
|   |   |-- platform_reboot
|   |   |-- pmon_daemon_control.json
|   |   |-- sensors.conf
|   |   |-- system_health_monitoring_config.json
|   |   |-- thermal_policy.json
```

- Files that are specific to a unique platform as recognized by ONIE but are shared among all hardware SKU variations of said platform should be placed directly under the appropriate `<ONIE_PLATFORM_STRING>/` directory.
- Files that are specific to a unique hardware SKU (port_config.ini, etc.) should be placed under the appropriate `<HARDWARE_SKU>/` directory within the appropriate `<ONIE_PLATFORM_STRING>/` directory. The hardware SKU is determined at boot time in the `default_sku` file.
- Files that are specific to a unique hardware SKU (buffer_defaults_t0/t1.j2) sets the ingress and egress buffer pool and buffer profile which is vendor dependent and would vary across different ASIC types.
- Files that are specific to a unique hardware SKU (pg_profile_lookup.ini) specifies the xon, xoff, size, threshold for different cable length connecting to the port operates at various speed levels.
- Files that are specific to a unique hardware SKU (Qos.json) specifies the QoS templates to be used for generating vendor specific settings which includes TC, PFC, DSCP,WRED etc.

- To support DPB (dynamic port breakout) feature, it is expected that we use same xxx.config.bcm for multiple hardware SKUs, the bcm file (example) is breakout capable so we can dynamically change the breakout modes at run time. To onboard the DPB feature, we would need define platform.json for the platform and hwsku.json files for hardware SKUs. See example and later sections for details.

Common Device-Specific File Details

- **default_sku**
    - Configuration file to setup default `<HARDWARE_SKU>` and topology.
    - Example:

```
Force10-S6000 t1
```

Force10-S6000 is the default hardware sku and t1 is the default topology.

- **fancontrol**
    - Configuration file for system fans used by fancontrol daemon.
        - NOTE: This file is deprecated. We suggest you utilize the SONiC Thermal Control Daemon (thermalctld) if you require a software-based solution to control your fans.
    - Format: Specified in the fancontrol documentation
    - Example:

```
INTERVAL=10
DEVPATH=hwmon2=devices/pci0000:00/0000:00:1f.3/i2c-0/0-002f
DEVNAME=hwmon2=w83795adg
FCTEMPS=hwmon2/device/pwm2=hwmon2/device/temp2_input
hwmon2/device/pwm1=hwmon2/device/temp2_input
FCFANS=hwmon2/device/pwm2=hwmon2/device/fan8_input
hwmon2/device/pwm2=hwmon2/device/fan7_input
hwmon2/device/pwm2=hwmon2/device/fan6_input
hwmon2/device/pwm2=hwmon2/device/fan5_input
hwmon2/device/pwm1=hwmon2/device/fan4_input
hwmon2/device/pwm1=hwmon2/device/fan3_input
hwmon2/device/pwm1=hwmon2/device/fan2_input
hwmon2/device/pwm1=hwmon2/device/fan1_input
MINTEMP=hwmon2/device/pwm2=20 hwmon2/device/pwm1=20
MAXTEMP=hwmon2/device/pwm2=60 hwmon2/device/pwm1=60
MINSTART=hwmon2/device/pwm2=75 hwmon2/device/pwm1=75
MINSTOP=hwmon2/device/pwm2=22 hwmon2/device/pwm1=22
```

    - Testing/Validation: Testing is device-specific. You will need to devise a way to test your temperature thresholds and related fan speeds.

- **installer.conf**
    - Configuration file for ONIE installer; allows for configuration of console device, port and speed as well as appending to the kernel command line
    - Requirements:
        - Must configure console port appropriately
    - Format: Key/value pairs in the form `<KEY>=<VALUE>`, one per line
        - `VAR_LOG_SIZE` (unit MB) can change the default VAR LOG SIZE to 100MB.
    - Example:

```
CONSOLE_PORT=0x2f8
CONSOLE_DEV=1
CONSOLE_SPEED=9600
ONIE_PLATFORM_EXTRA_CMDLINE_LINUX="acpi_enforce_resources=lax acpi=noirq"
VAR_LOG_SIZE=100
```

    - Testing/Validation: Build and install a SONiC image to confirm the installer was configured as expected.

- **platform_asic**
    - A data file which defined the ASIC vendor fact about the platform
    - Should contains lines and each line is a ASIC vendor name, such as broadcom/broadcom-dnx/mellanox/etc. It actually supported multiple lines with multiple vendor names, which is extremely seldom.
    - Example

```
broadcom-dnx
```

    - You can get a list of the ASIC platforms by `ls -b platform | cat`. Currently the options are

```
barefoot
broadcom
cavium
centec
```

```
centec-arm64
generic
innovium
marvell
marvell-arm64
marvell-armhf
mellanox
nephos
p4
vs
```

Also support

```
broadcom-dnx
```

- **platform_env.conf**
    - Configuration file to specify the environment parameters for platform modules.
        - NOTE: This file is optional. Needs to be present if platform needs to override default parameters for its modules.
    - Format: Key/value pairs in the form `<key>=<value>`, one per line
    - Example:

```
dmasize=128M
usemsi=1
```

    - Testing/Validation: Build and install a SONiC image to confirm the platform modules were loaded with the values configured.

- **led_proc_init.soc**
    - Initialization file for Broadcom LED microprocessors
        - NOTE: This file is optional. If your platform does not contain Broadcom LED microprocessors, this file may be omitted
    - Example:
        - https://github.com/sonic-net/sonic-buildimage/blob/master/device/dell/x86_64-dell_s6000_s1220-r0/led_proc_init.soc

    - Testing/Validation: Upon system restart, ensure LED microprocessors are initialized properly

- **platform.json**
    - A data file which defines static facts about the platform
    - Should contain a `chassis` element which in turn contains a hierarchy of platform hardware as is represented by the platform API. This section of the file will be utilized by the sonic-mgmt repository when performing regression tests on the platform API to ensure the data returned aligns with these facts.
    - Should contain an `interfaces` element which in turn contains information about the breakout modes available for each interface
        - Latest design doc for DPB (dynamic port breakout) and platform.json.
    - Examples:
        - x86_64-mlnx_msn2700-r0
        - x86_64-cel_seastone-r0

- **platform_reboot**
    - Executable script which communicates with platform-specific hardware to perform a "hard"/"forced" reboot by basically cycling power to the device
    - This script can be written in either Python or Bash; the choice is up to the vendor
    - Must be executable, and must be able to run without specifying an interpreter on the command line (i.e., it should execute by calling `./platform_reboot`)
    - Purpose is to write to appropriate hardware registers to cause the device to perform a hard reboot.
    - Note that this script should also take appropriate actions (if necessary) to ensure that this reboot is not detected as a hardware-caused reboot on the next boot; this should still be considered a user-initiated software reboot

- **pmon_daemon_control.json**

    - JSON file providing a platform-specific list of daemons inside the PMon container that should be skipped during bootup, more details are available in the design doc.
        - NOTE 1: This file is **NOT mandatory** if no daemons need to be skipped on the platform.
    - Format: One key/value pair for one daemon, a key is composed by 'skip' and daemon name, like 'skip_xcvrd', the value shall be 'true' if you want to skip launching the daemon. If the value is set to false, then the daemon will be launched, the behavior will be the same if the key/value not defined.
    - Example:

```
{
    "skip_ledd": true
}
```

    - Testing/Validation: Build a SONiC image and check the pmon container superviosrd conf file and 'start.sh' to see whether the skipped daemons have already been ruled out from the files, and to check whether the daemons are skipped/launched as expected after system bootup.

- **sensors.conf**
    - Libsensors configuration file. Used to configure sensor output from sensord daemon.
    - Requirements:
        - Provide clear and understandable labels for each sensor
        - Define critical values for each sensor. These can be defined either in this file or in hardware, as long as alarms are generated
    - Format: Specified in the sensors.conf documentation
    - Example:

```
# libsensors configuration file
# ------------------------------------------
#

bus "i2c-2" "SCD SMBus master 0 bus 0"
bus "i2c-3" "SCD SMBus master 0 bus 1"
bus "i2c-5" "SCD SMBus master 0 bus 3"
bus "i2c-6" "SCD SMBus master 0 bus 4"
```

```
bus "i2c-7" "SCD SMBus master 0 bus 5"

chip "k10temp-pci-00c3"
    label temp1 "Cpu temp sensor"

chip "lm73-i2c-3-48"
    label temp1 "Rear Temp Sensor"
    set temp1_max 65
    #set temp1_max_alarm 75 # read-only

chip "lm86-i2c-2-4c"
    label temp1 "Board Temp Sensor"
    set temp1_max 65
    set temp1_crit 75

    label temp2 "Front-panel Temp Sensor"
    set temp2_max 65
    set temp2_crit 75

chip "pmbus-i2c-3-4e"
    label temp1 "Power Controller Sensor 1"
    set temp1_max 60
    set temp1_crit 70

    label temp2 "Power Controller Sensor 2"
    set temp2_max 60
    set temp2_crit 70

    ignore curr1

chip "pmbus-i2c-5-58"
    label temp1 "Power Supply 1 Sensor 1"
    set temp1_max 60
    set temp1_crit 70

    label temp2 "Power Supply 1 Sensor 2"
    set temp2_max 60
    set temp2_crit 70

    ignore temp3

    set in1_max 250
    set in1_crit 255
    set power1_max 525
```

```
    set power2_max 460
    set power2_crit 462
    set curr1_max 5.28
    set curr1_crit 5.30
    set curr2_max 36
    set curr2_crit 37

chip "pmbus-i2c-6-58"
    label temp1 "Power Supply 2 Sensor 1"
    set temp1_max 60
    set temp1_crit 70

    label temp2 "Power Supply 2 Sensor 2"
    set temp2_max 60
    set temp2_crit 70

    ignore temp3

    set in1_max 250
    set in1_crit 255
    set power1_max 525
    set power2_max 460
    set power2_crit 462
    set curr1_max 5.28
    set curr1_crit 5.30
    set curr2_max 36
    set curr2_crit 37
```

        - Testing/Validation: Testing is device-specific. You will need to adjust your thresholds and confirm that alarms are generated accordingly.

- **pcie.yaml**
    - YAML file which contains information about the PCIe devices expected on the platform
    - Should contain the pcie `bus`, `dev`, `fn`, `id` and `name` information of all PCIe devices in the platform that need to be monitored.
    - The file will be utilized by the pcied in sonic-platform-daemons repository when monitoring the PCIe devices' status periodically and pcie-check.service which is checking the initial PCIe device status.

```
- bus: '00'
  dev: '14'
  fn: '2'
```

```
id: 1f41
name: 'Ethernet controller: Intel Corporation Ethernet Connection I354'
```

    - Examples (both are currently works-in-progress, and thus are incomplete):
        - https://github.com/sonic-net/sonic-buildimage/blob/master/device/dell/x86_64-dell_s6100_c2538-r0/pcie.yaml
        - https://github.com/sonic-net/sonic-buildimage/blob/master/device/arista/x86_64-arista_7060_cx32s/pcie.yaml
- This file can be generated using `pcieutil generate`.

- **hwsku.json**
    - A JSON file which specifies the default breakout modes for ports of the particular hardware SKU. (design and example)
    - Along with platform.json, we can deprecate below port_config.ini file.

- **port_config.ini [Note: Deprecated in favor of the use of platform.json and hwsku.json]**
    - Text file providing a SKU-specific mapping between SONiC port names, ASIC lanes and vendor port aliases.
    - Format: One line per port; variable number of columns, each separated by whitespace. Column order is also variable and *must* be specified by the column titles in the mandatory first line comment, as described below:
        - **name**: SONiC port name (required - Naming convention is at your discretion)
        - **lanes**: Comma-delimited list of the ASIC lanes connected to port (required)
        - **alias**: Vendor's alias for port (optional - this is helpful if converting an active switch from OEM software to SONiC. If the device was previously responding to SNMP queries using vendor port names, adding this alias will allow the device to continue responding to the SNMP queries). If alias column is omitted, SONiC will use port name as alias.

- **speed**: Port speed in megabits per second (Mb/s). E.g., 1Gb/s: `1000`, 10Gb/s: `10000`, 40Gb/s = `40000`, 100Gb/s = `100000`, etc.
- **autoneg**: Auto-negotiation. Valid values include:`0` (disabled), `1` (enabled)
- **fec**: Forward Error Correction (FEC). Valid values include: `none` (no FEC), `rs` (Reed-Solomon error correction)
- **index**: Physical index of port (optional - needed if ports do not begin with index zero or increment by one), When this column is missing the index starts from 0 and increase by 1 for each row. It is recommend to explicitly set this index column and make the index number match the labeling of the front panel ports. AKA: if the front panel ports are starting from 1, the index should start from 1. If the front panel ports start from 0, the index should start form 0. The break out ports from the same physical port should have same index.

- Example:

```
# name        lanes           alias         speed    autoneg   fec    index
Ethernet0     0,1,2,3         Ethernet0     100000   0         none   0
Ethernet4     4,5,6,7         Ethernet4     100000   0         none   1
Ethernet8     8,9,10,11       Ethernet8     100000   0         none   2
Ethernet12    12,13,14,15     Ethernet12    100000   0         none   3
Ethernet16    16,17,18,19     Ethernet16    100000   0         none   4
Ethernet20    20,21,22,23     Ethernet20    100000   0         none   5
Ethernet24    24,25,26,27     Ethernet24    100000   0         none   6
Ethernet28    28,29,30,31     Ethernet28    100000   0         none   7
Ethernet32    32,33,34,35     Ethernet32    100000   0         none   8
Ethernet36    36,37,38,39     Ethernet36    100000   0         none   9
Ethernet40    40,41,42,43     Ethernet40    100000   0         none   10
Ethernet44    44,45,46,47     Ethernet44    100000   0         none   11
Ethernet48    48,49,50,51     Ethernet48    100000   0         none   12
Ethernet52    52,53,54,55     Ethernet52    100000   0         none   13
Ethernet56    56,57,58,59     Ethernet56    100000   0         none   14
Ethernet60    60,61,62,63     Ethernet60    100000   0         none   15
Ethernet64    64,65,66,67     Ethernet64    100000   0         none   16
Ethernet68    68,69,70,71     Ethernet68    100000   0         none   17
Ethernet72    72,73,74,75     Ethernet72    100000   0         none   18
Ethernet76    76,77,78,79     Ethernet76    100000   0         none   19
Ethernet80    80,81,82,83     Ethernet80    100000   0         none   20
Ethernet84    84,85,86,87     Ethernet84    100000   0         none   21
Ethernet88    88,89,90,91     Ethernet88    100000   0         none   22
Ethernet92    92,93,94,95     Ethernet92    100000   0         none   23
```

```
Ethernet96    96,97,98,99       Ethernet96    100000   0         none    24
Ethernet100   100,101,102,103   Ethernet100   100000   0         none    25
Ethernet104   104,105,106,107   Ethernet104   100000   0         none    26
Ethernet108   108,109,110,111   Ethernet108   100000   0         none    27
Ethernet112   112,113,114,115   Ethernet112   100000   0         none    28
Ethernet116   116,117,118,119   Ethernet116   100000   0         none    29
Ethernet120   120,121,122,123   Ethernet120   100000   0         none    30
Ethernet124   124,125,126,127   Ethernet124   100000   0         none    31
```

    - Testing/Validation: TODO

- **sai.profile / Broadcom config file**
    - Profile for initializing SAI / Broadcom ASIC configuration file
    - Format: Key/value pairs in the form `<KEY>=<VALUE>`, one per line
    - NOTE: Platforms based on Broadcom ASICs require a specific **config.bcm** file which is specified in the `sai.profile` file by the `SAI_INIT_CONFIG_FILE` property.
        - Please make sure sai.profile contents are as follows: `SAI_INIT_CONFIG_FILE=/usr/share/sonic/hwsku/<your config.bcm>`
        - If sai.profile is generated by a j2 template, please make sure it points to the new config.bcm file under /usr/share/sonic/hwsku/
    - How to test new Broadcom configuration file:
        - Add new Broadcom configuration file to `device/<VENDOR_NAME>/<ONIE_PLATFORM_STRING>/<HARDWARE_SKU>` folder.
        - Run a regular make to build an image. If the build passes, the configuration file also passed the smoke test.
        - Install the built image on DUT and test it. The smoke test doesn't guarantee the configuration file actually works.
    - What if the new Broadcom configuration file contains entries denied by the build test?
        - Please don't create PR. It won't get approved or merged as-is.
        - Please don't add these entries to the permitted list. The PR won't be approved or merged.
        - Please contact Broadcom with list of entries currently being denied, ask Broadcom to give written permission to add these entries. Once we have the written approval from Broadcom, we will add the newly approved entries to the permission list. Then you can create PR.

- Examples:
  - device/arista/x86_64-arista_7260cx3_64/Arista-7260CX3-D108C8/sai.profile
  - device/arista/x86_64-arista_7260cx3_64/Arista-7260CX3-D108C8/th2-a7260cx3-64-112x50G+8x100G.config.bcm

- **led_control.py - Note: Will be deprecated and replaced by the SONiC Platform API Package in the future**
  - Python module providing a unified API for updating front-panel LED state. It is used by the SONiC LED control daemon.
    - NOTE: This file is optional. If your device uses an an entirely hardware-based LED control solution you can omit this file for your device.
  - Format: Python module that must implement the class `LedControl` which must inherit from `sonic_led.led_control_base.LedControlBase`. This class receives notifications when changes in link state occur and must update front panel LED state according to your platform-specific implementation.
  - Example:
    - https://github.com/sonic-net/sonic-buildimage/blob/master/device/arista/x86_64-arista_7050_qx32/plugins/led_control.py
  - Testing/Validation:
    - Disconnect all cables from QSFP ports. All link status LEDs should indicate a "link down" state.
    - Enter the Redis shell on the switch by running `redis-cli`
    - Manually toggle link states in the Redis database and ensure the link status LEDs react accordingly. For example:
      - To set the link state for port Ethernet24 to "up": `127.0.0.1:6379> hset PORT_TABLE:Ethernet24 oper_status up`
      - To set the link state for port Ethernet24 to "down": `127.0.0.1:6379> hset PORT_TABLE:Ethernet24 oper_status down`

- **system_health_monitoring_config.json**

    - JSON file providing a way to customize system health monitoring daemon configuration. more details are available in the design doc.
        - NOTE 1: This file is **NOT mandatory** if no customization needed for the system health monitoring daemon on the platform.
    - Format: Key/value pairs in the form `<KEY>:<VALUE>`, one per line.
    - Example:
        - https://github.com/sonic-net/sonic-buildimage/blob/master/device/mellanox/x86_64-mlnx_msn2700-r0/system_health_monitoring_config.json

- **thermal_policy.json**
    - JSON file which including all the thermal policies for the thermal control daemon on the platform. more details are available in the design doc.
        - NOTE 1: This file is **NOT mandatory** if no thermal policy defined on the platform.
    - Format: Key/value pairs in the form `<KEY>:<VALUE>`.
    - Example:
        - https://github.com/sonic-net/sonic-buildimage/blob/master/device/mellanox/x86_64-mlnx_msn2700-r0/thermal_policy.json