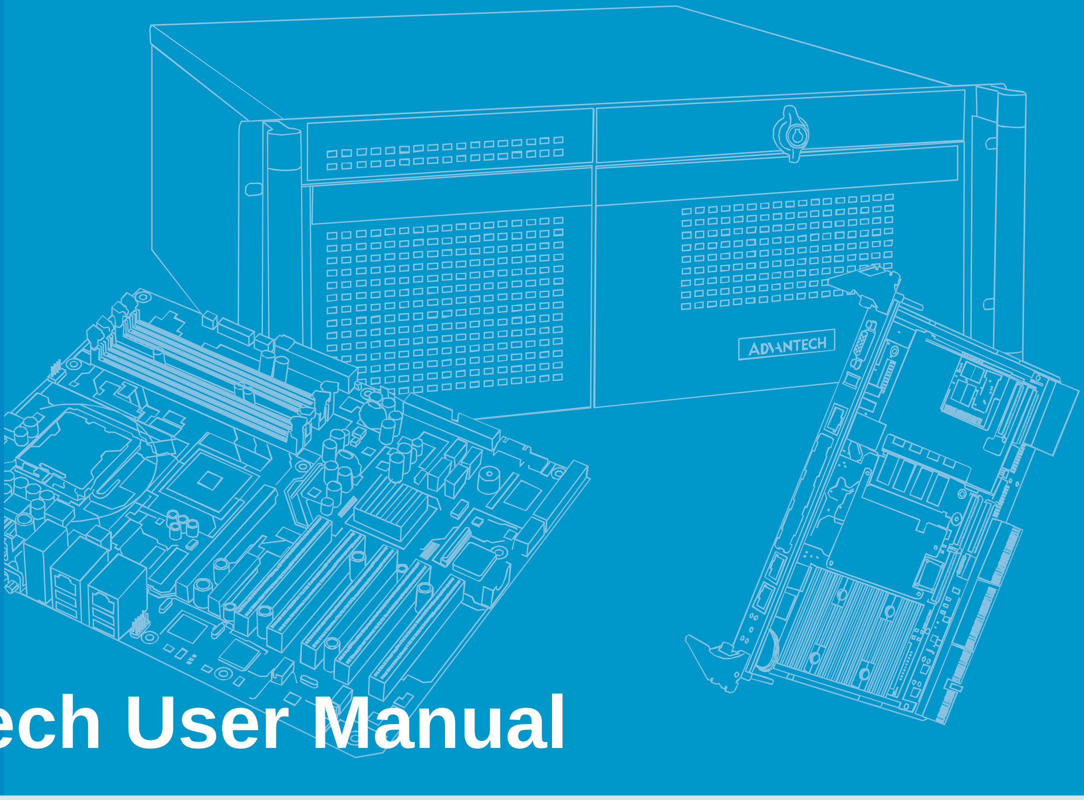

Advantech User Manual

Advanced LAN Bypass

Edition 7

Jan. 2023

Enabling an Intelligent Planet

Copyright

Warnings, Cautions and Notes

Warning! *Warnings indicate conditions, which if not observed, can cause personal injury!*

Caution! *Cautions are included to help you avoid damaging hardware or losing data. e.g.*

There is a danger of a new battery exploding if it is incorrectly installed. Do not attempt to recharge, force open, or heat the battery. Replace the battery only with the same or equivalent type recommended by the manufacturer. Discard used batteries according to the manufacturer's instructions.

Note! *Notes provide optional additional information.*

We Appreciate Your Input

Please let us know of any aspect of this product, including the manual, which could benefit from improvements or corrections. We appreciate your valuable input in helping make our products better.

Please send all such - in writing to: ncg@advantech.com

Revision

Date [mm/dd/yyyy]	Revision	Modifications
1/24/2023	7.0	Official release edition 7
12/23/2022	6.01	- Modify master slave mode related description. Add “slave” option if BMC is the master. - Add method that use apdi –lbpcu.cfg to handle I2C Bus number changing.
05/21/2022	6.0	Official release edition 6
01/20/2022	5.01	Change I801 to SYS_I2C to cover AMD/HYGON platform.
11/23/2021	5.0	Added Advanced bypass I2C support.
07/14/2020	4.01	Added note in 1.1 Scope about using GLIBC and mmap().
06/25/2020	4.0	Official release edition 4
06/22/2020	3.1	1. Added SLA note for liblbpcu. 2. Updated 1.1 Scope table.
09/25/2019	3.00	Official release edition 3 1. added 3.7.2.6 lbp_get_customized_id(unsigned char *id) 2. added 4.44 ./lbpcu –LoadDefault
05/23/2019	2.00	Official release edition 2 for LBPCU v1.30 (Liblbpcu v1.26)
09/20/2018	1.00	Official release edition 1

Contents

Tables

Figure

Chapter 1 Advanced LAN Bypass

1.1 Scope

This document describes Advantech Advanced LAN Bypass and related SW APIs.

It covers the following SW and FW releases (See 3.6 for more details):

Table 1.1 Scope

Release date	(NIC) FW version	(I2C) FW version	Liblbpcu version	LBPCU version	Protocol version
2016/02/04	00.22		00.16	00.12	00.06
2016/06/28	01.02		01.00	01.00	00.06
2017/08/21	01.14		01.12	01.12	00.08
2017/09/04	01.16		01.12	01.14	00.08
2017/10/31	01.18		01.14	01.16	00.08
2018/05/10	01.20		01.16	01.18	00.08
2018/07/06	01.20		01.18	01.20	00.08
2018/09/03	01.20		01.20	01.22	00.08
2018/11/08	01.20		01.20	01.24	00.08
2018/12/05	01.20		01.22	01.26	00.08
2019/01/14	01.20		01.24	01.28	00.08
2019/05/13	01.20		01.26	01.30	00.08
2019/09/30	01.22		01.28	01.32	00.08
2019/12/30	01.22		01.30	01.34	00.08
2020/03/30	01.22		01.32	01.36	00.08
2020/06/02	01.24		01.34	01.38	00.08
2021/10/05	01.24	01.00	02.00	02.00	00.10
2022/1/20	01.24	01.00	02.02	02.02	00.10
2022/12/09	01.24	01.00	02.04	02.04	00.10

Note: *If you want to use LBPCU with kernel 2.6.x, please use LBPCU 1.26 or later version.*

LBPCU and Liblbpcu are both built with GLIBC and using mmap(), so the customer's Linux OS must support GLIBC and mmap(), otherwise LBPCU and Liblbpcu will not work. If you meet some problem using LBPCU, please check Appendix A for different Linux distributions. If you are using KVM or other virtual machine, please check Appendix B for more information.

Please check the release date on the Advantech website or contact with your Advantech representative for details supported product and SW version.

1.2 Terminology

Table 1.2 Terminology

Term	Description
Action	A specific action that the Advanced LAN bypass mechanism should take for a specific event
API	Application programming interface
BMC	Board Management Controller
Bypass segment	One bypass segment consists of two (adjacent) network ports connected through bypass relays (electrical or optical)
Event	A system event (e.g., power up, power down, reset, WDT timeout)
FW	Firmware
FlowNIC	Advantech's intelligent NIC series for flow processing/load balancing on high-end network appliances
Host system	x86 system
I2C	Inter-Integrated Circuit.
LAN Bypass MCU	LAN Bypass micro controller
LBP	LAN Bypass
LBPCU	Advanced LAN bypass control utility
LED	Light emitting diode
Liblbpcu	Library for Advanced LAN Bypass
NIC	Network interface controller
NMC	Network mezzanine card (an Advantech proprietary form factor NIC card)
PCIe	PCI Express
PFA	PCI function address
PMM	PHY mezzanine module
WDT	Watchdog timer

1.3 Advanced LAN Bypass Hardware Architecture Overview

Advanced LAN Bypass may be supported on Ethernet ports hosted either directly on a motherboard as onboard network ports or via add-in cards such as NMCs or PCIe NIC cards.

The NICs are connected to the CPU (s) via the PCIe interface.

A dedicated and buffered power rail allows the LAN bypass mechanism to detect system events (e.g., power outages or resets) and take the appropriate predefined action.

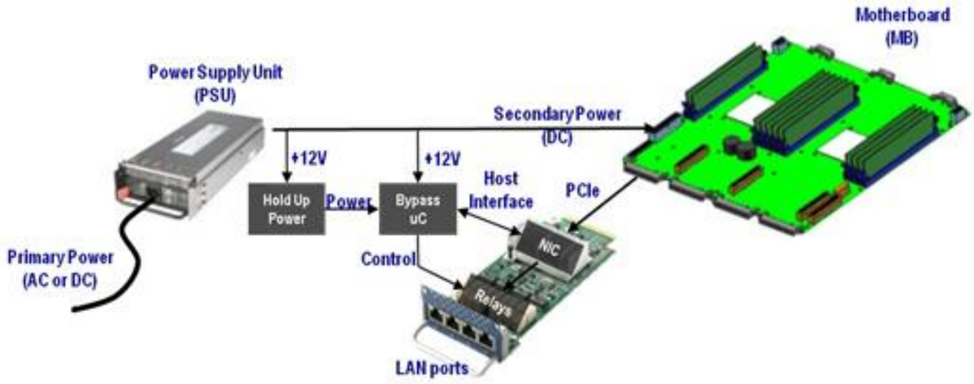

Figure 1.1 LAN Bypass Hardware Overview

Two network ports form one bypass segment. The individual segments on the platform are implicitly defined through hardware design (i.e., signal and relay connectivity).

Bypass control happens on a per-segment basis.

Both network ports of a segment can either connect to their associated NICs, be disconnected from their associated NICs, or be switched to bypass mode where both of the physical LAN ports connect directly to each other, isolating them from their associated NICs (See section 2.3 for more details).

A LAN Bypass MCU controls this connectivity by switching optical or electrical relays between the ports. Latching relays are used to guarantee that the bypass state or connectivity is maintained during an absence of power.

Using the Advanced LAN Bypass library and related SW APIs, applications running on the host system can access the LAN Bypass MCUs in the system in order to change bypass behavior or to activate and use additional features such as the integrated WDT functionality.

The interface between the host SW and the LAN Bypass MCUs are optimized for low-latency operation. The LAN Bypass library is designed as a user space library and communicates directly with the NICs.

At 2021, Advantech add new capability to Advanced LAN Bypass to support centralized control, called Advanced LAN Bypass (I2C.) And original Advanced LAN Bypass called Advanced LAN Bypass (NIC).

Advanced LAN Bypass (I2C) use Host or BMC I2C as host interface. And there are some minor behavior differences between Advanced LAN Bypass (NIC) and Advanced LAN Bypass (I2C). Which will be covered in the following section.

Chapter 2 Overview

2.1 Host Interface

The Advantech Advanced LAN Bypass (NIC) mechanism uses a direct low-latency connection between the NIC(s) and the LAN Bypass MCU(s).

The Advantech Advanced LAN Bypass (I2C) uses Host or BMC I2C connection between host and the LAN Bypass MCU(s).

2.2 Segment Control

Bypass control is always performed at the segment level; that is, each bypass segment can be controlled independently and uses distinct resources such as bypass WDT.

This allows various schemes for applications controlling the bypass segments.

In most cases, a centralized control application or OS is used to manage LAN bypass on the system. In such a scenario, the control plane can still control each bypass segment independently (e.g., force one segment into bypass mode but keep another segment connected).

Moreover, WDTs may be used independently or be centrally controlled by using the global WDT trigger feature. For more details regarding supported WDT mechanisms, refer to Section 2.5.

For Advanced LAN Bypass (NIC,) it is also possible that several OS or application instances control their respective bypass segment(s) independently, instead of using a single, global control method. In this case, each application or instance only accesses the "local" NIC(s) and LAN Bypass MCU(s). This eliminates shared resources which would create dependencies between these instances via spin locks or mutexes.

When a specific NIC and LAN Bypass MCU are used by one application instance, the other application instances cannot access that NIC and LAN Bypass MCU. In this case, lbpcu will return “the segment is locked.” If multiple application instances need to access the same segment or same LAN Bypass MCU, the suggestion is to create one management application instance as a broker to access the segment or LAN Bypass MCU. The other application instances can access Advanced LAN Bypass through this management application instance instead of accessing the segment and LAN Bypass MCU directly. It will avoid the mutex lock.

For Advanced LAN Bypass (I2C), Prefer architecture will be one OS or management application instance to control all segments in the system. Different OS or application instances to control different segment in one system or different application instances trying to access I2C bus will cause I2C bus conflict. Spin locks or mutexes are used to avoid different application that use liblbpcu to access the I2C buses, but utility that is not using liblbpcu is covered.

The global WDT trigger feature can still be used for synchronization between bypass segments in this mode.

2.3 Event and Action

Advanced LAN Bypass uses an event driven architecture to provide maximum flexibility to define the LAN bypass behavior of each segment.

The LAN Bypass MCU will perform the specified action for each of the platform events.

There are eight events supported by Advanced LAN Bypass:

- **Power Up**
  The host system is turned on/powers up
- **Power Down**
  The host system is turned off/powers down or there is a power failure
- **Power Reset**
  The host system is reset or rebooted
- **WDT Start**
  LAN Bypass WDT is started or strobed for the first time
- **WDT Timeout**
  LAN Bypass WDT times out
- **External Trigger**
  Global WDT trigger input (dedicated GPIO pin)
- **Button Event_1**
  The master button (supported on specific platforms) has been pushed
- **Button Event_2**
  The master button (supported on specific platforms) has been pushed again

There are four actions supported by Advanced LAN Bypass:

CONNECT
Connect the bypass segment to the host system (i.e., both network ports of the segment connect to the respective NICs/host)

DISCONNECT
Disconnect the bypass segment (i.e., both network ports of the segment are isolated from the respective NICs/host and from each other)

BYPASS
Connect both ports in the bypass segment, while it isolates them from the respective NICs/host (i.e., network traffic is possible between both ports, but bypasses the host itself)

DO_NOT_CHANGE
Means that the state will not be changed when a specified event occurs (i.e., network connectivity will remain as is).

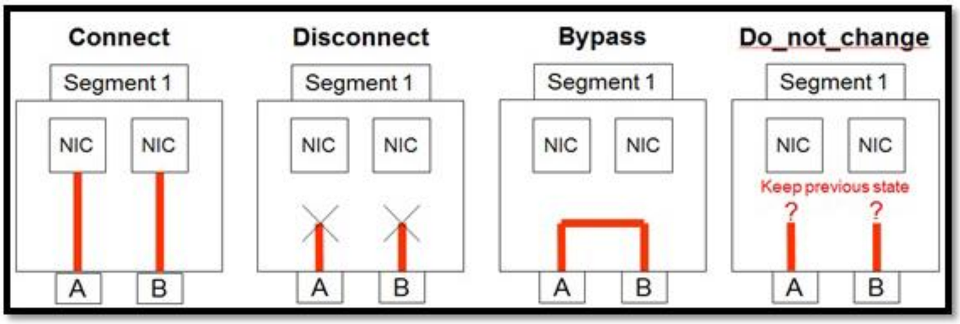

Figure 2.1 Bypass Actions and Related Port Connectivity

The default action setting for all events (not include POWER DOWN) is "CONNECT", which means LAN Bypass is turned off. The default action setting for POWER DOWN event is “DISCONNECT.”

Example 1:

- Connect the LAN ports to the host system when the system is powered and operational
- Bypass the NICs in case of power failure or in case application SW hangs
- Maintain the states of network connectivity (no change) in case of a system reset
- Master/slave mode is disabled

Table 2.1 Mapping between Events and Actions (Example 1)

Event	CONNECT	DISCONNECT	BYPASS	DO_NOT_ CHANGE
Power Up	X			
Power Down			X	
Power Reset				X
WDT Start	X			
WDT Timeout			X	
External Trigger				X
Button Event 1				X
Button Event 2				X

Example 2:

- Connect the LAN ports to the host system ONLY when application SW is running and strobing the WDT
- Bypass the NICs if the power is down or application SW hangs
- Disconnect the LAN ports from the NICs in other states (system is booting, application is not yet loaded)
- Master/slave mode is disabled

Table 2.2 Mapping between Events and Actions (Example 2)

Event	CONNECT	DISCONNECT	BYPASS	DO_NOT_ CHANGE
Power Up		X		
Power Down			X	
Power Reset		X		
WDT Start	X			
WDT Timeout			X	
External Trigger		X		

Button Event 1				X
Button Event 2				X

Example 3:

- Connect the LAN ports to the host system only when application SW is running and strobing the WDT
- Do not allow any traffic through the unit or bypassing the unit in any other situation
- Master/Slave mode is disabled

Table 2.3 Mapping between Events and Actions (Example 3)

Event	CONNECT	DISCONNECT	BYPASS	DO_NOT_ CHANGE
Power Up		X		
Power Down		X		
Power Reset		X		
WDT Start	X			
WDT Timeout		X		
External Trigger		X		
Button Event 1				X
Button Event 2				X

Example 4:

- Connect the LAN ports to the host system only when application SW is running and strobing the WDT
- Do not allow any traffic through the unit or bypassing the unit in any other situation
- Set all segments to the same behavior (master/slave mode is enabled)
- A button is required in the front or rear panel that can:
  - Connect all ports when pushed by the user at the first time.
  - Bypass all ports when pushed by the user again.
  - Each push will make the event switched between button event 1 and button event 2.

Table 2.4 Mapping between Events and Actions (Example 4)

Event	CONNECT	DISCONNECT	BYPASS	DO_NOT_ CHANGE
Power Up		X		
Power Down		X		
Power Reset		X		
WDT Start	X			
WDT Timeout		X		

External Trigger			X
Button Event 1	X		
Button Event 2		X	

2.4 Manual Bypass Control

In addition to the event-driven model, it is also possible to force an action on a specific LAN bypass segment via the Advanced LAN Bypass SW API. For example, an application might enable previously disconnected network ports once it starts up and is ready to process data. The last event will show up as "Manual" to distinguish it with the Event & Action list in Section 0.

2.5 Watchdog Timer

Each segment supports an independent WDT. The timer base is set to 100 ms. The WDT timeout period can be set between 100 ms and 6553.5 s (approximately 109 min).

Several WDT modes are supported. In addition to the standard WDT mode, Advanced LAN Bypass also supports an additional feature called "toggle pin mode," which reduces the communication latency and overhead between the NIC and LAN Bypass MCU. It also allows automatic WDT start-up.

Both mechanisms are supported in parallel, meaning that regular WDT control is still available even though toggle pin mode is enabled.

Meanwhile, both mechanisms are used to strobe the same WDT. WDT expiration will always trigger a WDT timeout event, regardless of the WDT having been previously strobed through the toggle pin or a SW API.

2.5.1 Standard Watchdog Mechanism

In this mode, the WDT in the LAN Bypass MCU is controlled by the host SW via the regular messaging interface. The host SW issues API commands to configure and start, stop, or reset the WDT (See 3.8.4 for more details).

The WDT is also strobed (to keep it from timing out) via the messaging interface (inband).

2.5.2 Toggle Pin Mechanism

The Toggle Pin related function is for Advanced LAN Bypass (NIC) only.

In this mode, a dedicated signal ("toggle pin") between the NIC and LAN Bypass MCU is used to strobe the WDT and keep it from timing out. Every signal state change (i.e., each rising or falling edge) will be treated as a WDT strobe.

This mechanism is disabled by default to provide backward compatibility with the Legacy LAN Bypass implementation. To enable the toggle pin mechanism, two activation methods are supported:

- It can be activated "on demand" by issuing the respective command via the regular messaging interface between the host and LAN Bypass MCU
- It can be permanently enabled by configuration and a specific start sequence on the toggle pin will start the timer

In toggle pin mode, no communication over the regular messaging interface is required. The host SW simply keeps toggling the dedicated signal in order to strobe the WDT.

Arming and auto re-arming functionalities further simplify the use of the toggle pin mode.

2.5.2.1 On Demand Mode

Once toggle pin mode is enabled and set to "on demand" (manual) mode, the WDT is armed.

Once armed, the host SW needs to execute a specific start sequence on the toggle pin (initiated through the "lbp_start_wdt_by_toggle" API call) to start the WDT.

In normal operation, the WDT will be disarmed when it expires and needs to be re-armed by the host SW.

For cases where manual re-arming is not desired, auto re-arming can be enabled. When this is enabled, the WDT will be automatically re-armed when it expires. A subsequent change of the toggle signal will restart the WDT.

The auto re-arm function can be configured as a non-volatile setting of the LAN Bypass MCU.

2.5.2.2 Permanent Mode

Toggle pin mode can be permanently enabled as a non-volatile configuration setting for the LAN Bypass MCU. In this mode, toggle pin mode will automatically be enabled and the WDT will be armed by default.

In addition, the auto re-arming features can be used as in "on demand" mode.

In combination, the non-volatile configuration setting for the WDT count, toggle pin permanent enable, and auto re-arm allow for a one-time configuration of the WDT via the host messaging interface. Once configured, no inband messaging is required anymore but all WDT operations can be accomplished out-of-band via the toggle pin.

Toggle pin mode summary:

- Enable/disable (volatile setting)
  Toggle pin mode manual enable/disable (on demand mode)
- Arm/disarm (volatile setting)
  Arm/dis-arm the WDT (if armed, a change in toggle signal will implicitly start the WDT)
- Auto re-arm (non-volatile setting)
  When the WDT expires, it will be re-armed by a subsequent change of the toggle signal
- Arm and enable (non-volatile setting)
  Toggle pin mode will be automatically enabled and the WDT will be armed by default (i.e., upon FW start-up)

2.5.3 Global Watchdog Trigger Mechanism

Advanced LAN Bypass supports global synchronization between multiple bypass segments through a hardware signal. In this mode, a master segment may activate the trigger signal and the other LAN Bypass MCUs will receive the signal as slaves. Reception of the trigger signal will cause the external trigger event if the segment is configured as a slave. In this case, the master as well as the slave LAN Bypass MCUs will perform the defined action for the external trigger event.

Either one single or multiple masters can assert the global WDT trigger signal.

Any LAN Bypass MCU can be configured to be a master or slave or to ignore the global WDT trigger signal via dedicated Advanced LAN Bypass SW API commands. (Advanced LAN Bypass (I2C) LEGACY HW version can support slave or ignore only)

Certain Advantech hardware platforms also support other masters such as the Baseboard Management Controller (BMC) or the x86 chipset (PCH). Please refer to the related hardware platform documentation for details.

Two modes are supported for triggering the event: edge trigger and level trigger.

In edge trigger mode, a falling edge of the hardware signal will trigger an external trigger event. The bypass segment will be switched to the corresponding state as defined by the external trigger event action and remain in this state until another event occurs or until a manual action is received via an API. Advantech suggests using edge trigger mode for Advanced LAN Bypass (NIC) and Advanced LAN Bypass (I2C).

In level trigger mode, a low-level hardware signal will trigger an external trigger event. The bypass segment will be switched to the corresponding state as defined by the external trigger event action, and it will remain in this state as long as the hardware signal stays asserted. Other events will be ignored as long as the global WDT trigger signal remains at a low level. When the hardware signal de-asserts (goes high), the bypass segment will return to the previous state that was active before assertion of the signal. If you want to de-assert the global WDT pin, please use -RW command on the master segment. Otherwise the slave segments can't switch to any other event and will receive a NACK error message when the user tries to reset the WDT of the slave segments.

Global WDT trigger mode summary:

- Disabled
  Global WDT trigger is not enabled.
- Enable, slave mode, edge trigger
  Global WDT trigger is enabled in slave mode and uses edge trigger.
- Enable, slave mode, level trigger
  Global WDT trigger is enabled in slave mode and uses level trigger.
- Enable, master mode, edge trigger
  Global WDT trigger is enabled in master mode and uses edge trigger.
- Enable, Master mode, Level trigger
  Global WDT trigger is enabled in master mode and uses level trigger.

NOTE: All settings for the global WDT trigger are non-volatile and will be preserved across power cycles or in the event of power failure.

Examples:

- Segment 1 is configured as a master and Segments 2 and 3 are configured as slaves.
  - When the Segment 1 WDT expires, it will trigger an external trigger event on Segments 2 and 3.
  - Segment 1 will perform the action defined for the WDT timeout event, while Segments 2 and 3 will perform the action defined for the external trigger event.
- Segments 1 and 2 are configured as masters and Segment 3 is configured as a slave.
  - When either the Segment 1 or 2 WDT times out, it will trigger an external trigger event on Segment 3.
  - Segment 1 or 2 will perform the action defined for the WDT timeout event, while Segment 3 will perform that defined for the external trigger event.

2.6 Bypass Master/Slave Mode with Button and LED Control

The purpose of master/slave mode is to treat the entire system as one bypass segment. It is not to be confused with the master/slave mode supported by the global WDT trigger mechanism described in section 2.5.3 Global Watchdog Trigger Mechanism.

One segment in the configuration needs to be defined as a master, and that master handles the **Master/Slave Mode** button and **Master/Slave Mode** LED. The rest of the segments defined in lbpcu.cfg will act as slaves.

The state of the master segment will always be propagated to the slave segments so that all segments in the system share the same state.

The **Master/Slave Mode** button and **Master/Slave Mode** LED are based on the previous assumption of treating the entire system as one bypass segment.

Master/Slave Mode must be enabled to make the **Master/Slave Mode** button and **Master/Slave Mode** LED work.

You can use the **Master/Slave Mode** button to trigger Button Event_1 and Button Event_2 on all segments, and to check what action is taken on the system, as indicated by the **Master/Slave Mode** LED.

Master/slave mode is supported on all platforms and NMCs. Button control and the **Master/Slave Mode** LED, however, are supported on specific platforms that have a **Master/Slave Mode** button and **Master/Slave Mode** LED (e.g., FWA-6520L, FWA-6170. Please refer to the product specifications). User can set actions for Button Event_1 and Button Event_2 on the system without a **Master/Slave Mode** button or **Master/Slave Mode** LED, but these two events will not be triggered because there is no button to press.

When master/slave mode is enabled, the external trigger or global WDT mechanism will be disabled and vice versa.

2.6.1 Master

The master can be one specific segment or BMC.

When the master is one specific segment, the master segment is triggered by the **Master/Slave Mode** button and controls the **Master/Slave Mode** LED. When the **Master/Slave Mode** button is pushed, the master segment will perform the preconfigured action and sync the state to all slave segments.

When the master is BMC, the BMC will handle the **Master/Slave Mode** LED and button. In this case, all segments in the system act as slaves. User need to use IPMI related command to configure **Master/Slave Mode** LED and button.

There can be only one master in the system.

NOTE: Only Advanced LAN Bypass (NIC) support Master mode.

2.6.2 Master/Slave Mode LED

The **Master/Slave Mode** LED shows the system's bypass **state** based on the **Master/Slave Mode** LED configuration.

NOTE: Only Advanced LAN Bypass (NIC) supports Master/Slave Mode LED.

2.6.3 Master/Slave Mode Button

The **Master/Slave Mode** button can be used to directly change the bypass state. It alternatingly triggers the events Button Event 1 and Button Event 2.

When the button is pushed for the first time, Button Event 1 will be triggered. When the button is pushed again, Button Event 2 will be triggered. Another push results in another trigger of Button Event 1, and so on.

NOTE: Only Advanced LAN Bypass (NIC) supports Master/Slave Mode Button.

Warning! *The* Master/Slave Mode button is non-operational in the power-off state!

2.7 LED Indication

Advantech Advanced LAN Bypass uses the **Activity** LED located on the right side of the LAN connector to indicate the current LAN Bypass state of each segment/port.

Table 2.5 Activity LED Behavior

State	LED Status	Comment
CONNECT	Green on	Blinking when there is any network activity (regular activity LED function)
BYPASS	Solid amber	
DISCONNECT	Blinking amber	1 Hz

Table 2.6 Link Speed LED Behavior

Link Speed	LED Status	Comment
NO LINK	Off	Check the Activity LED to distinguish "NO LINK" and "10M"
10M	Off	
100M	Amber	
1G	Green	

2.7.1 Adjustable Blinking Frequency for Bypass/Disconnect States

The blinking frequency for bypass or disconnect states can be adjusted by changing the **Duty** and **Period** parameters through the Advanced LAN Bypass SW API.

NOTE: The blinking frequency setting for bypass and disconnect is based on the specific MCU that controls the segment. This means that if two segments are controlled by the same MCU, then the blinking frequency setting will be shared by the two segments.

Duty can be set as an integer between 0 and 100. The value represents the duty cycle as a percentage.

Period can be set between 1 and 255 (increment: 100 ms). The LAN Bypass MCU uses the following formula to calculate T_{ON} (ON time) and T_{OFF} (OFF time).

$$T_{ON}=(\text{Period} \times 100) \times (\text{Duty} / 100)\ (\text{ms})$$

$$T_{OFF}=(\text{Period} \times 100) \times (1-\text{Duty}/100)\ (\text{ms})$$

Examples:

1. Duty = 30, Period = 1 (blinking every 100 ms). TON/TOFF will be 30/70 ms.

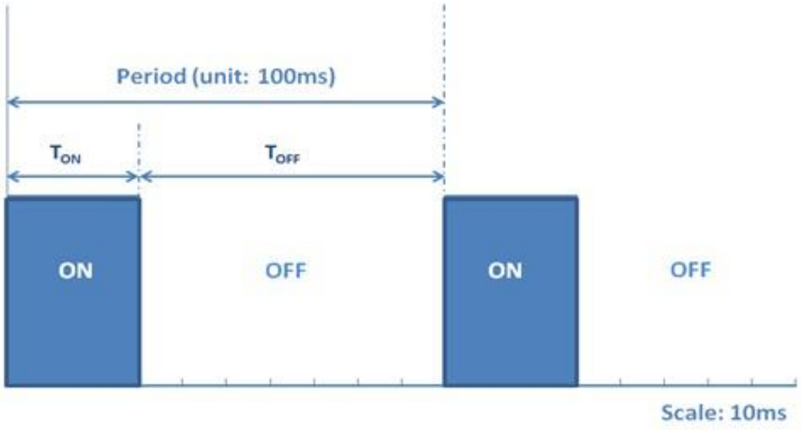

Figure 2.2 Adjustable Blinking Frequency

2. Duty = 0 means that the LED is always OFF (disable LED)
3. Duty = 100 means that the LED is always ON (solid amber, default setting for the BY-PASS state)
4. Duty = 50, Period = 10 (blinks every 1 s). TON/TOFF will be 500/500 ms (default setting for the DISCONNECT state)

2.8 Bypass control type

Advanced LAN Bypass can control Copper type or Fiber type bypass design. Since the control mechanism are different, Advantech will set correct type during manufacturing.

2.9 Advanced LAN Bypass I2C HW version

Advanced LAN Bypass (I2C) has two HW version. Advantech will set correct type during manufacturing.

- LEGACY: HW design based on Legacy LAN Bypass.
- NEW: HW design based on Advanced LAN Bypass (I2C).

2.10 Error Codes

Table 2.7 Error Code

Error Code	Name	Comment
-128	LBP_ERR_OPEN_FILE	Failed to open the file, cannot open the file (either the configuration file or the upgrade image), or the file does not exist
-127	LBP_ERR_CONF_FILE_FORMAT	Configuration file format error; please check the configuration file
-126	LBP_ERR_CONF_GET_PCI_DEVICE	Cannot retrieve the PCI device ID definition from the configuration file; please check the configuration file
-125	LBP_ERR_CLOSE_FILE	Failed to close the file
-124	LBP_ERR_PCI_BASE_ADDR	Cannot find the specified MCU on the PCIe interface; please check the configuration of PCIe devices in the configuration file
-123	LBP_ERR_DEV_MEM_OPEN	Failed to access physical memory; please check the user permission level
-122	LBP_ERR_MMAP	Memory mapping failure
-121	LBP_ERR_GBASE_OPEN	Failed to access registers for GPIO access; please check the PCIe device configuration
-120	LBP_ERR_LBP_CLOSE	Failed to close lbp because interfaces were busy (timeout)
-119	LBP_ERR_INTERFACE_OPEN	Failed to open the interface (timeout)
-118	LBP_ERR_FW_NRDY	The FW is not ready or the FW is busy (timeout)
-117	LBP_ERR_FW_CS	FW checksum is invalid
-116	LBP_ERR_DATA_FORMAT	Erroneous file format of the FW upgrade image; please check the upgrade image file
-115	LBP_ERR_GET_FW_MODE	Cannot enter FW mode
-114	LBP_ERR_UNLOCK_FLASH	Cannot unlock the flash
-113	LBP_ERR_SWITCH_TO_BOOTLOADER	Cannot switch to the boot loader
-112	LBP_ERR_START_TRANSFER	Cannot start the transfer
-111	LBP_ERR_SWITCH_TO_APP	Cannot switch to the application
-110	LBP_ERR_COMMUNICATION	FW upgrade communication failure
-109	LBP_ERR_PARAM_OUT_OF_BOUND	Parameter is out of range; please check the correctness of the input parameter
-108	LBP_ERR_PARAM_NULL_POINTER	Null pointer parameter is specified; please check the API call for correctness
-107	LBP_ERR_LBP_INIT	LBP initialization check error; the LBP library needs to be initialized first before any other function calls
-106	LBP_ERR_COMMAND_NOT_SUPPORT	Command unsupported; please check whether the command is supported on the current FW protocol version (FW update required?)
-105	LBP_ERR_GPIO_SET	Failed to set GPIO

-104	LBP_ERR_DEVICE_TYPE	Unknown device type found in the configuration file; please check the configuration file
-103	LBP_ERR_NFP_CONNECTION	Failed to open socket to NFP from the x86 side
-102	LBP_ERR_NFP_SYNC	Data transfer incomplete
-101	LBP_ERR_NFP_ACCESS	Cannot access NFP
-100	LBP_ERR_NFP_PARAMETER	NFP parameter error
-32	LBP_ERR_NACK_CMD	Invalid command
-31	LBP_ERR_NACK_PARAM	FW parameter is out of range
-30	LBP_ERR_NACK_CS	Checksum mismatch
-29	LBP_ERR_NACK_CURRENT_STATE	Cannot perform the action in the current state

Chapter 3 liblbpcu – Advanced LAN Bypass library

Advantech provides a Linux library (liblbpcu.a) and corresponding header file (liblbpcu.h) so you can integrate Advanced LAN Bypass control into your applications.

NOTE: The customer needs to sign Software License Agreement (SLA) to get "Linux library (liblbpcu.a) and corresponding header file (liblbpcu.h)", kindly contact Advantech representative for SLA initiation.

3.1 Supported Distributions

liblbpcu is a generic Linux user space library and thus should work with most Linux distributions.

Advantech uses up-to-date Ubuntu distributions to perform SW quality assurance and release testing. For more details, please refer to the release notes delivered as part of LBPCU.

3.2 The lbpcu.cfg File

Per library instance, one configuration file (default name: lbpcu.cfg) is used to define the bypass segment(s) and related resources to be controlled by this specific instance. Please make sure the application instance and the lbpcu.cfg are in the same path (folder). LBPCU will search the same path of binary first. If not found, then LBPCU will search the working path. If you want to change the name or path of lbpcu.cfg or use APIs, then please set the variable in cmd_ret_var.h.

```
char* g_config_path
```

Basically, the configuration file consists of one or multiple segment definitions.

Using different configuration files for different segments allows for the use of parallel library instances (e.g., individual applications or application instances controlling individual network segments). For LBPCU version < 1.30, please make sure you only use 1 application instance to access 1 segment. We can't use multi-processes or multi-threads to control a single LAN bypass segment. Multi-processes trying to access a single segment at the same time will cause communication errors and unexpected results.

Since LBPCU version 1.30, we support multi-processes and multi-threads to access the same segment. But please note that only one application can access the segment and LAN Bypass MCU in the same time. Please check chapter 2.2 for more information.

The configuration file is a plain text file and consists of one or multiple blocks that start with the keyword "SEGMENT" followed by a segment number. This number is user-defined and can be anything in the range of 0~255, as long as each number is unique to each configuration file.

For Advanced LAN Bypass (I2C), suggest to use one application instance only to avoid I2C bus conflict.

While the numbering scheme is local to each liblbpcu instance, in practical use, segment numbers typically follow the physical ports definition on a platform for ease of use. Within the SEGMENT block, the physical access parameters for the specific segment are defined. These are:

- PCIe BUS:DEV:FUN in case of Advanced LAN Bypass (NIC) (NMC or PCIe NIC)
  Please see section 3.3 for further details on the exact assignment.
- CHIP:SITE:SEG in case of FlowNIC
- TYPE:BUS:MCU_SEGMENT in case of Advanced LAN Bypass (I2C)
  Please see section 0 for further details on the exact assignment.

We've added a new option in the configuration file since LBPCU version 1.30.

Users can now add

```
eth_name: eth0
```

before

```
SEGMENT: 1
```

This option allows the user to use "eth" name instead of segment number in LBPCU commands. Please check more information in chapter 4. This option only works with LBPCU version >=1.30.

A sample lbpcu.cfg file for two bypass segments:

```
eth_name: eth0
SEGMENT: 1
BUS: 5e
DEV: 0
FUN: 0

eth_name: eth1
SEGMENT: 2
BUS: 5e
DEV: 0
FUN: 2

===== DO NOT CHANGE THE FORMAT IN THIS FILE =====

This file is the configuration setting for LBPCU segment
If TYPE is PCIE
BUS: (hexadecimal)
DEV: (hexadecimal)
FUN: (hexadecimal)
If TYPE is FLOWNIC
CHIP: (hexadecimal)
SITE: (hexadecimal)
SEG:   (hexadecimal)
If TYPE is I2C
TYPE: (BMC.X.IPMI   / SYS_I2C, X in BMC type is to determine the /dev/ipmiX. lbpcu
          will use IPMI Master Write-Read command if the final parameter set as
          "IPMI". If platform use PCH I2C, use " SYS_I2C".)
BUS: (Decimal Root_Port.I2C_SWITCH_1_CHANNEL.I2C_SWITCH_2_CHANNEL, 255
means
         do not have this switch.)
MCU_SEGMENT: (Decimal 1-4)
```

3.3 Segment Control for PCIe-based Devices

NOTE: The following information is only required to understand the mechanism and configuration file. The configuration file needs to be set properly before using LBPCU or the API.

Regardless of whether dedicated single-port NICs or higher density (dual/quad port) NICs are being used, the convention is that a segment will always be controlled from the first port of the segment.

For example:

- A NMC has one dual-port NIC with PFAs 03:00.0 and 03:00.1.
  The control port (messaging interface) is 03:00.0, so the parameters for this segment in the configuration file would be:

– BUS: 3

– DEV: 0

– FUN: 0

- A NMC has one quad-port NIC with PFAs 03:00.0 ~ 03:00.3.
  The control port is 03:00.0 for the first segment (consisting of NICs 03:00.0 and 03:00.1) and 03:00.2 for the second segment (consisting of NICs 03:00.2 and 03:00.3). In the configuration file, the parameters for the first segment would be:

– BUS: 3

– DEV: 0

– FUN: 0

Those for the second segment would be:

– BUS: 3

– DEV: 0

– FUN: 2

- A mainboard has six single-port NICs with PFAs 01:00.0 ~ 06:00.0.
  The control port is 01:00.0 for the first segment (consisting of NICs 01:00.0 and 02:00.0), 03:00.0 for the second segment (consisting of NICs 03:00.0 and 04:00.0), and 05:00.0 for the third segment (consisting of NICs 05:00.0 and 06:00.0).

3.4 Segment Control for I2C-based Devices

NOTE: The following information is only required to understand the mechanism and configuration file. The configuration file needs to be set properly before using LBPCU or the API.

Since I2C related information do not have clear relationship with PCIe NIC device, so user need to identify the Host information, I2C bus, I2C switch channels and MCU segment number for each segment.

When there is BMC in the platform, there are two methods to set configuration.

- Use major I2C bus and control the I2C switch manually.
- Use extend I2C bus that is using BMC I2C switch driver to handle the I2C switch.
  This will require BMC to extend the bus number in IPMI Master write-read command. Please check with Advantech representative if your platform supports this function. The response time is faster than controlling I2C switch manually because lbpcu doesn't need to set I2C switch.

When a platform does not have BMC, the user can choose if they want to use the PCA9548 driver to handle the I2C switch. The user can check the following example with and without a PCA9548 driver.

- TYPE: Represents the host information,
  - SYS_I2C: represents the Host is x86 PCH I2C bus.
  - BMC: represents the Host is BMC I2C bus.
    - X: represents which /dev/ipmiX to use to communicate with BMC
    - IPMI: represents which command to use.
      - IPMI: Uses IPMI Master Write-Read command.
- BUS: Represents which I2C bus, first and second I2C switch channel settings.
  - First digit: Represents I2C bus number. (0-255)
  - Second digit: Represents first I2C switch channel (0-7.) 255 means you do not have this switch or it is not needed.
  - Third digit: Represents second I2C switch channel (0-7.) 255 means you do not have this switch or it is not needed.
- MCU_SEGMENT: Represent the Segment number on MCU. (Decimal 1 ~ 4)

For example:

- Segment ID 6 is on a platform that has BMC, uses /dev/ipmi1 (second KCS channel) to access BMC, and uses IPMI Master Write-Read to access MCU. If the BMC bus is 6, the first I2C switch channel is 2, the second I2C switch is not used, and MCU segment is 2
  - SEGMENT: 6
  - TYPE: BMC.1.IPMI
  - BUS: 6.2.255
  - MCU_SEGMENT: 2
- Segment ID 7 is on a platform with BMC, uses /dev/ipmi0 (First KCS channel) to access BMC, and use IPMI Master Write-Read to access MCU. If the BMC extend bus is 10, instead of lbpcu, BMC I2C switch driver will handle the first and second I2C switch, and MCU segment is 4
  - SEGMENT: 7

- TYPE: BMC.0.IPMI
- BUS: 10.255.255
- MCU_SEGMENT: 4

- Segment ID 2 is on a platform that does not have BMC and uses SYS_I2C, the SYS_I2C bus is 0, first I2C switch channel is 4, the second I2C switch channel is 0, and MCU segment is 1

- SEGMENT: 2
- SYS_I2C
- BUS: 0.4.0
- MCU_SEGMENT: 1

- Segment ID 0 is on a platform that does not have BMC and use SYS_I2C, the user loads PCA9548 driver to handle the I2C switch. If MCU is at PCA9548 bus 24, MCU segment is 3

- SEGMENT: 0
- SYS_I2C
- BUS: 24.255.255
- MCU_SEGMENT: 3

3.5 Header File

The header file exports/provides common definitions for the API commands. A brief introduction is provided here for reference only. Please check the header file for more details:

Note! *The definition of Error Codes can be found in 0.*

```
/* Events */
#define DEFAULT                          0
#define POWER_ON                         1
#define POWER_OFF                        2
#define POWER_RST                        3
#define WATCHDOG_START                   4
#define WATCHDOG_TIMEOUT                 5
#define EXTERNAL_TRIGGER                 6
#define MANUAL                           7
#define BUTTON_EVENT_1                   8
#define BUTTON_EVENT_2                   9

/* Action*/
#define BYPASS                           1
#define DISCONNECT                       2
#define CONNECT                          3
#define DO_NOT_CHANGE                    0xFF

/* Macros for WDT status */
/* Those macros are used to check watchdog timer status and the returned value to each macro is either TRUE or FALSE */
/* cf. Sections 2.5 and 3.7.4)
#define lbp_is_wdt_running(x)            ((x & 0x01) == 0x01) //WDT is running
#define lbp_is_wdt_ext_en(x)             ((x & 0x04) == 0x04) //External (Global) WDT is enabled
#define lbp_is_wdt_ext_master(x)         ((x & 0x06) == 0x06) //External WDT is enabled in Master mode

#define lbp_is_wdt_ext_edge(x)         ((x & 0x80) == 0x00) //External WDT is enabled in Edge mode#define lbp_is_wdt_ext_level(x)              ((x & 0x80) == 0x80) //External WDT is enabled in Level mode
#define lbp_is_wdt_ext_slave(x)        ((x & 0x06) == 0x04) //External WDT is enabled in Slave mode
#define lbp_is_wdt_toggle_en(x)        ((x & 0x08) == 0x08) //Toggle pin WDT is enabled as On Demand mode
#define lbp_is_wdt_toggle_armed(x)     ((x & 0x10) == 0x10) //Toggle pin WDT is armed
#define lbp_is_wdt_toggle_rearm_en(x) ((x & 0x20) == 0x20) //Toggle pin WDT is auto-rearmed
#define lbp_is_wdt_toggle_en_default(x) ((x & 0x40) == 0x40) //Toggle ping WDT is permanentlly enabled and automatically armed by default (e.g. upon firmware starts up)
```

```
/* Macros for Reset Monitoring State. */
#define lbp_is_rmt_running(x)                    ((x & 0x01) == 0x01)
#define RM_ignore(x)                            ((x & 0x01)==0x00)
#define SRM_NA(x)                                ((x & 0x06)==0x00)
#define SRM_by_BIOS(x)                           ((x & 0x06)==0x02)
#define SRM_by_LBPCU(x)                          ((x & 0x06)==0x04)
#define SRM_by_timeout(x)                        ((x & 0x06)==0x06)

/* Macros for MCU FW upgrade mode */
/* Those macros are used to check firmware upgrade status and the returned value to each macro is
either TRUE or FALSE */
#define lbp_is_in_ap(x)                          ((x & 0x10) == 0x10) //FW is running in application model, not
in upgrade mode
#define lbp_is_in_bootloader(x)              ((x & 0x20) == 0x20) //FW is running in upgrade mode
#define lbp_is_bl_idle(x)((x & 0x2f) == 0x20) //FW upgrade process is started
#define lbp_is_bl_write_complete(x)       ((x & 0x2f) == 0x21) //FW upgrade is on going
#define lbp_is_bl_flash_error(x) ((x & 0x2f) == 0x22) //FW upgrade fails
#define lbp_is_bl_wrong_data_format(x)((x & 0x2f) == 0x23) //Fail to receive upgrade image file

/* Mode for External watchdog. */
#define EXT_WDT_DISABLE_MODE              0x00
#define EXT_WDT_SLAVE_MODE                0x01 /* Trigger wdt expired ACTION when wdt_RST is
low. */
#define EXT_WDT_MASTER_EDGE_MODE   0x02 /* Drive wdt_RST to low when Watchdog expires.
*/
#define EXT_WDT_MASTER_LEVEL_MODE   0x12

/* Mode for Toggle pin wdt. */
#define TOGGLE_ENABLE_PIN            1           <<0       //bit 0: Enable (1) or disable (0) toggle pin
#define TOGGLE_ARM_PIN                 1           <<1       //bit 1: Arm (1) or disarm (0) toggle pin
#define TOGGLE_AUTO_REARM         1             <<2       //bit 2: Automatically rearm (1) or don't re-
arm (0) when watchdog counter expires



#define TOGGLE_ARM_AND_ENABLE      1    <<3       //bit 3: "Arm and enable" (1) or "don't

enable and arm" (0) by default.
```

3.6 API Command List

The following table lists all available API commands and highlights the required FW/library (liblbpcu) versions and details the protocol version for debugging:

Table 2.8 Support API functions

API	Section	Supported FW version (NIC)	Supported FW version (I2C)	Supported liblbpcu version	Supported protocol version
lbp_check_conf_file	3.8.1.1	00.04	01.00	00.04	00.01
lbp_open	3.8.1.2	00.04	01.00	00.04	00.01
lbp_close	3.8.1.3	00.04	01.00	00.04	00.01
lbp_get_version	3.8.2.1	00.04	01.00	00.04	00.01
lbp_get_num_segments	3.8.2.2	00.04	01.00	00.04	00.01
lbp_get_fw_version	3.8.2.3	00.04	01.00	00.04	00.01
lbp_get_protocol_version	3.8.2.4	00.04	01.00	00.04	00.01
lbp_get_action	3.8.3.1	00.04	01.00	00.04	00.01
lbp_set_action	3.8.3.2	00.04	01.00	00.04	00.01
lbp_set_wdt_counter	3.8.4.1	00.04	01.00	00.04	00.01
lbp_get_wdt_status	3.8.4.2	00.04	01.00	00.04	00.02
lbp_reset_start_wdt	3.8.4.3	00.04	01.00	00.04	00.01
lbp_stop_wdt	3.8.4.4	00.04	01.00	00.04	00.01
lbp_set_toggle_pin_mode	3.8.5.1	00.04	N/A	00.04	00.01
lbp_set_ext_wdt*	3.8.4.5	00.04	01.00	00.04	00.02
lbp_nic_wdt_toggle	3.8.5.3	00.04	N/A	00.04	00.01
lbp_get_last_event_action	3.8.3.3	00.04	01.00	00.04	00.01
lbp_set_current_action	3.8.3.4	00.04	01.00	00.04	00.01
lbp_check_upgrade_image	3.8.6.1	00.04	01.00	00.04	00.01
lbp_get_fw_mode	3.8.6.2	00.04	01.00	00.04	00.01
lbp_upgrade_fw	3.8.6.3	00.04	01.00	00.04	00.01
lbp_start_reset_monitoring	3.8.7.1	00.06	01.00	00.06	00.03
lbp_get_reset_monitoring_state	3.8.7.2	00.06	01.00	00.06	00.03
lbp_set_rmt_counter	3.8.7.3	00.06	01.00	00.06	00.03
lbp_get_rmt_status	3.8.7.4	00.06	01.00	00.06	00.03
lbp_start_wdt_by_toggle	3.8.5.2	00.06	N/A	00.06	00.03
lbp_set_message_level	3.8.8.1	00.10	01.00	00.08	00.04
lbp_set_bypass_ctrl_type	3.8.9.1	00.10	01.00	00.08	00.04
lbp_get_bypass_ctrl_type	3.8.9.2	00.10	01.00	00.08	00.04
lbp_get_bl_fw_version	3.8.2.5	00.16	01.00	00.10	00.05

lbp_set_bypass_led	3.8.11.1	00.18	01.00	00.12	00.06
lbp_get_bypass_led	3.8.11.2	00.18	01.00	00.12	00.06
lbp_set_disconnect_led	3.8.11.3	00.18	01.00	00.12	00.06
lbp_get_disconnect_led	3.8.11.4	00.18	01.00	00.12	00.06
lbp_set_master_slave_mode*	3.8.12.1	01.10	01.00	01.10	00.08
lbp_get_master_slave_mode	3.8.12.2	01.10	01.00	01.10	00.08
lbp_set_master_led	3.8.12.3	01.10	N/A	01.10	00.08
lbp_get_master_led	3.8.12.4	01.10	N/A	01.10	00.08
lbp_scan_event_init	3.8.14.1	01.16	01.00	01.14	00.08
lbp_scan_event_detach	3.8.14.2	01.16	01.00	01.14	00.08
lbp_check_FW	3.8.15.1	01.16	01.00	01.22	00.08
lbp_eth_name_to_seg_num	3.8.1.4	01.16	01.00	01.26	00.08
lbp_get_advanced_bypass_type	3.8.2.7	N/R	N/R	02.00	N/R
lbp_set_advanced_bypass_i2c_hw_version	3.8.10.1	N/A	01.00	02.00	00.10
lbp_get_advanced_bypass_i2c_hw_version	3.8.10.2	N/A	01.00	02.00	00.10
lbp_reset_eeprom	3.8.13.1	N/A	01.00	02.00	00.10

N/R: Not related.

N/A: Command do not support.

lbp_set_ext_wdt*: Advanced LAN Bypass (I2C) HW version LEGACY cannot support MASTER_EDGE or MASTER_LEVEL mode.

lbp_set_master_slave_mode*: Advanced LAN Bypass (I2C) cannot support Master rule.

3.7 How to Use the API

The following flow summarizes how to use the LBP library:

- Call the lbp_open function to allocate and initialize resources
- Perform functions or operations on the bypass segment(s)
- Call the lbp_close function to properly release the allocated resources

3.8 API Functions

3.8.1 LBPCU Open/Close

3.8.1.1 lbp_err lbp_check_conf_file (char * conf_file)

Checks the configuration file.

This function will automatically be called by lbp_open(). It is deprecated as a direct function call but is kept for backwards compatibility.

Returns 0 on success or the following error codes in the case of an error:

Table 3.1 API Function "lbp_close" Error Codes

Name	Error Code
LBP_ERR_OPEN_FILE	-128
LBP_ERR_CONF_FILE_FORMAT	-127
LBP_ERR_CONF_GET_PCI_DEVICE	-126
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.1.2 lbp_err lbp_open(int segment)

Opens the communication interface for a specified bypass segment.

This function needs to be called before executing any other function on the specific LAN bypass segment.

Returns 0 on success or the following error codes in the case of an error:

Table 3.2 API Function "lbp_open" Error Codes

Name	Error Code
LBP_ERR_OPEN_FILE	-128
LBP_ERR_CONF_FILE_FORMAT	-127
LBP_ERR_PCI_BASE_ADDR	-124
LBP_ERR_DEV_MEM_OPEN	-123
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_GPIO_SET	-105

3.8.1.3 lbp_err lbp_close(int segment)

Closes an opened communication interface of a specific LAN bypass segment.

Returns 0 on success or the following error codes in the case of an error:

Table 3.3 API Function "lbp_close" Error Codes

Name	Error Code
LBP_ERR_LBP_INIT	-107
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_CLOSE	-120

3.8.1.4 lbp_err lbp_eth_name_to_seg_num (char *conf_file, char *eth_name, char *seg_number)

Input conf_file and eth_name, then this function outputs the seg_number.

Returns 0 on success or the following error codes in the case of an error:

Table 3.4 API Function " lbp_eth_name_to_seg_num " Error Codes

Name	Error Code
LBP_ERR_OPEN_FILE	-128
LBP_ERR_CONF_FILE_FORMAT	-127
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.2 LBP Information

3.8.2.1 lbp_err lbp_get_version(unsigned char *major, unsigned char *minor)

Description:

Returns the major and minor versions of liblbpcu

Refers to the release notes for more information on liblbpcu versions.

Returns 0 on success or the following error code in the event of an error:

Table 3.5 API Function "lbp_get_version" Error Codes

Name	Error Code
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.2.2 lbp_err lbp_get_num_segments(int *segment_number)

Returns the total number of segments defined in the configuration file.

Returns 0 on success or the following error code in the event of an error:

Table 3.6 API Function "lbp_get_num_segments" Error Codes

Name	Error Code
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.2.3 lbp_err lbp_get_fw_version(int segment, unsigned char *major, unsigned char *minor)

Returns the FW version of the LAN Bypass MCU.

Refers to the release notes for more information on MCU FW versions.

Returns 0 on success or the following error codes in the event of an error:

Table 3.7 API Function "lbp_get_fw_version" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.2.4 lbp_err lbp_get_protocol_version(int segment, unsigned char *lib_major, unsigned char *lib_minor, unsigned char *fw_major, unsigned char *fw_minor)

Returns major and minor protocol versions of the liblbpcu and LAN Bypass MCU FW.

When a user application calls API functions that are available only on FW version 0.04 (protocol version 0.01 or later), liblbpcu will check whether the protocol version of the LAN Bypass MCU FW matches the requirement. If it does not match, the error code "LBP_ERR_COMMAND_NOT_SUPPORT" will be returned.

Refers to the release notes for more information on protocol versions.

Returns 0 on success or the following error codes in the event of an error:

Table 3.8 API Function "lbp_get_protocol_version" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.2.5 lbp_err lbp_get_bl_fw_version(int segment, unsigned char *major, unsigned char *minor)

Returns the bootloader version of the LAN Bypass MCU FW.

Returns 0 on success or the following error codes in the event of an error:

Table 3.9 API Function "lbp_get_bl_fw_version" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.2.6 lbp_get_customized_id(unsigned char *id)

Description:

Returns the customized id of liblbpcu

Returns 0 on success or the following error code in the event of an error:

Table 3.10 API Function "lbp_get_customized_id " Error Codes

Name	Error Code
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.2.7 lbp_get_advanced_bypass_type(int segment, int *advanced_bypass_type);

Description:

Returns the Advanced LAN Bypass type for specific segment.

```
/* Advanced LAN Bypass type. */
#define ADVANCED_BYPASS_NIC_TYPE            0x00
#define ADVANCED_BYPASS_FLOWNIC_TYPE      0x01
#define ADVANCED_BYPASS_I2C_TYPE            0x02
```

Returns 0 on success or the following error code in the event of an error:

Table 3.11 API Function " lbp_get_advanced_bypass_type " Error Codes

Name	Error Code
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.3 LBP Event/Action

3.8.3.1 lbp_err lbp_get_action(int segment, int event, int *action)

Returns the current action for the specified event on a specific segment.

Events:

```
#define POWER_ON                1
#define POWER_OFF               2
#define POWER_RST               3
#define WATCHDOG_START          4
#define WATCHDOG_TIMEOUT        5
#define EXTERNAL_TRIGGER        6
#define MANUAL                  7
#define BUTTON_EVENT_1          8
#define BUTTON_EVENT_2          9
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.12 API Function "lbp_get_action" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.3.2 lbp_err lbp_set_action(int segment, int event, int action)

Sets the current action for a specified event on a specific segment.

Events:

```
#define POWER_ON                1
#define POWER_OFF               2
#define POWER_RST               3
#define WATCHDOG_START          4
#define WATCHDOG_TIMEOUT        5
#define EXTERNAL_TRIGGER        6
#define MANUAL                  7
#define BUTTON_EVENT_1          8
#define BUTTON_EVENT_2          9
```

Action:

```
#define BYPASS                  1
```

```
#define DISCONNECT          2
#define CONNECT             3
#define DO_NOT_CHANGE       0xFF
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.13 API Function "lbp_set_action" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107

3.8.3.3 lbp_err lbp_get_last_event_action(int segment, int *last_event, int *last_action)

Returns the last event and the last action (i.e., the current state of bypass relays) on a specific segment.

Returns 0 on success or the following error codes in the event of an error:

Table 3.14 API Function "get_last_event_action" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.3.4 lbp_err lbp_set_current_action(int segment, int action)

Manually sets an immediate action on a specific segment.

This function can be used to perform a certain action on a specific segment (e.g., to switch the bypass relays into bypass mode manually without affecting any event-action definitions).

The "Do not change" action is not supported in this function.

Action:

```
#define BYPASS          1
#define DISCONNECT      2
#define CONNECT         3
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.15 API Function "lbp_set_current_action" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107

3.8.4 LBP Watchdog

3.8.4.1 lbp_err lbp_set_wdt_counter(int segment, int time_100ms)

Sets the initial countdown value (100 ms per count) of the WDT.

The WDT is a 16-bit counter with a maximum value of 65535.

The setting is non-volatile and will be used as the initial value whenever the WDT is started or restarted. The default value is 0x0100 (25.6 s).

Returns 0 on success or the following error codes in the event of an error:

Table 3.16 API Function "lbp_set_wdt_counter" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107

3.8.4.2 lbp_err lbp_get_wdt_status (int segment, int *wdt_status, int *init_time_100ms, int *current_time_100ms)

Retrieves the current WDT status.

Additional predefined macros (See 3.5) are available for evaluation of the returned value.

Returns 0 on success or the following error codes in the event of an error:

Table 3.17 API Function "lbp_get_wdt_status" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108

LBP_ERR_LBP_INIT	-107

3.8.4.3 lbp_err lbp_reset_start_wdt (int segment)

Resets the WDT counter to the initial countdown value (See section 3.8.4.1) and starts the timer.

Returns 0 on success or the following error codes in the event of an error:

Table 3.18 API Function "lbp_reset_start_wdt" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107

3.8.4.4 lbp_err lbp_stop_wdt(int segment)

Stops the WDT and sets the current timer value to 0.
Returns 0 on success or the following error codes in the event of an error:

Table 3.19 API Function "lbp_stop_wdt" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.4.5 lbp_err lbp_set_ext_wdt(int segment, int ext_wdt_mode)

Sets the global WDT trigger mode.
Advanced LAN Bypass (I2C) HW version LEGACY do not support MASTER_EDGE or MASTER_LEVEL
Returns 0 on success or the following error codes in the event of an error:

Table 3.20 API Function "lbp_set_ext_wdt" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109

LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

The following global WDT trigger modes are supported:

Table 3.21 Parameter ext_wdt_mode

Parameter ext_wdt_mode	Description
0x00	global trigger is disabled (default setting)
0x01	global trigger slave mode
0x02	global trigger master mode, edge triggered
0x03	global trigger master mode, level triggered

NOTE: Global WDT trigger mode is disabled by default. This setting is non-volatile and will be stored locally in EEPROM.

3.8.5 LBP Toggle

Advanced LAN Bypass (I2C) do not support Toggle function.

3.8.5.1 bp_err lbp_set_toggle_pin_mode(int segment, int toggle_mode)

Sets the toggle pin mode.

Returns 0 on success or the following error codes in the event of an error:

Table 3.22 API function "lbp_set_toggle_pin_mode" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

Toggle pin mode is set with bit masks defined as follows:

Table 3.23 Parameter toggle_mode

Parameter toggle_mode Bitmasks	Meaning
Bit 0 = 0b	Toggle mode is disabled (volatile)
Bit 0 = 1b	Toggle mode is set as on demand mode (volatile)
Bit 1 = 0b	Dis-arm WDT (volatile)
Bit 1 = 1b	Arm WDT (volatile)

Bit 2 = 0b	No auto re-arm (non-volatile)
Bit 2 = 1b	Auto re-arm (nonvolatile)
Bit 3 = 0b	Toggle mode (non-volatile) is not permanently enabled
Bit 3 = 1b	Toggle mode is permanently enabled and the WDT is automatically armed on startup (non-volatile)

The header file (See section 3.4) provides a few macros to simplify bit operations:

```
#define TOGGLE_ENABLE_PIN          1    <<0     //bit 0: Enable (1) or disable (0) toggle pin
#define TOGGLE_ARM_PIN             1    <<1     //bit 1: Arm (1) or disarm (0) toggle pin
#define TOGGLE_AUTO_REARM          1    <<2     //bit 2: Automatically rearm (1) or don't rearm (0)
when watchdog counter expires.
#define TOGGLE_ARM_AND_ENABLE  1    <<3    //bit 3: "Arm and enable" (1) or "don't enable and arm"
(0) by default.
```

3.8.5.2 lbp_err lbp_start_wdt_by_toggle(int segment)

Sends the WDT start sequence through the toggle pin interface.

Toggle pin mode needs to be enabled on this segment for this command to take effect.

Returns 0 on success or the following error codes in the event of an error:

Table 3.24 API function "lbp_start_wdt_by_toggle" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.5.3 lbp_err lbp_nic_wdt_toggle(int segment)

Toggles the toggle pin for a specified segment to strobe the WDT in toggle pin mode.

Please refer to 2.5.2 for details on toggle pin mode.

Toggle pin mode needs to be enabled and armed on this segment for this command to take effect.

Returns 0 on success or the following error codes in the event of an error:

Table 3.25 API function "lbp_nic_wdt_toggle"

Name	Error Code
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107

3.8.6 LBP FW Upgrade

3.8.6.1 lbp_err lbp_check_upgrade_image(char *image_file)

Checks the integrity of the FW upgrade image.

This function validates whether the FW upgrade image is compatible with the LAN Bypass MCU before starting the FW upgrade.

Returns 0 on success or the following error codes in the event of an error:

Table 3.26 API function "lbp_check_upgrade_image" Error Codes

Name	Error Code
LBP_ERR_OPEN_FILE	-128
LBP_ERR_DATA_FORMAT	-116
LBP_ERR_PARAM_NULL_POINTER	-108

3.8.6.2 lbp_err lbp_get_fw_mode(int segment, int *mode)

Returns the current FW upgrade status.

This function is intended for debugging purposes during FW upgrade.

Returns 0 on success or the following error codes in the event of an error:

Table 3.27 API function "lbp_get_fw_mode" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

These header files (seeSee section 3.4 Header File) provide the following macros, which are used at different FW upgrade stages:

```
/* Macros for Upgrade MODE */
#define lbp_is_in_ap(x)                ((x & 0x10) == 0x10) //FW is running in application
model, not in upgrade mode
#define lbp_is_in_bootloader(x)        ((x & 0x20) == 0x20) //FW is running in upgrade mode
#define lbp_is_bl_idle(x)              ((x & 0x2f) == 0x20) //FW upgrade process is started
#define lbp_is_bl_write_complete(x)    ((x & 0x2f) == 0x21) //FW upgrade is on going
#define lbp_is_bl_flash_error(x)       ((x & 0x2f) == 0x22) //FW upgrade fails
```

```
#define lbp_is_bl_wrong_data_format(x)      ((x & 0x2f) == 0x23) //Fail to receive upgrade image file
```

3.8.6.3 lbp_err lbp_upgrade_fw(int segment, char *image_file)

Upgrades the LAN Bypass MCU FW.

A character pointer (parameter image_file) is used to specify the image file name (with the file path). For example, a user application may perform a FW upgrade on Segment 1 by using the following function call:

```
lbp_upgrade_fw(1, "./lbp_fw_RU_00_22.bin")
```

The file operation will be performed internally in the function.

Returns 0 on success or the following error codes in the event of an error:

Table 3.28 API Function "lbp_upgrade_fw" Error Codes

Name	Error Code
LBP_ERR_OPEN_FILE	-128
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_DATA_FORMAT	-116
LBP_ERR_GET_FW_MODE	-115
LBP_ERR_UNLOCK_FLASH	-114
LBP_ERR_SWITCH_TO_BOOTLOADER	-113
LBP_ERR_START_TRANSFER	-112
LBP_ERR_SWITCH_TO_APP	-111
LBP_ERR_COMMUNICATION	-110
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107

3.8.7 LBP Reset Monitoring

On standard x86 platforms, a reset signal may be triggered by system components during system booting. This action may be taken by BIOS or by the chipset to reconfigure the PCIe bus. As such, LAN Bypass MCU will treat this reset as a new event and will perform the defined action for a POWER RESET event.

However, it may cause unexpected and undesirable behavior. For example, when the system is powered on, the last event should be POWER ON when the system boots into the OS. However, if the BIOS/chipset resets the platform during boot up, the last event would be POWER RESET, which is not the expected event after powering on the system.

Advanced LBP will disable the reset monitoring during power on. The reset monitor will start if the "start reset monitoring" command is issued or if the reset monitoring timer times out.

The default reset monitoring timer is set to 60s.

Some use cases are provided here for reference:

- Sets the reset monitoring timer to a very high value and let BIOS send the "start reset monitoring" command (See 3.8.7.1).
- Sets the reset monitoring timer to 0, LAN Bypass MCU will start the reset monitor immediately after power on/reset.
- Sets the reset monitoring timer to a specific value and skip the reset during system boot up.

3.8.7.1 lbp_err lbp_start_reset_monitoring(int segment)

Forces LAB Bypass MCU to start monitoring reset signal immediately.

(See 3.8.7 for more information.)

Returns 0 on success or the following error codes in the event of an error:

Table 3.29 API Function "lbp_start_reset_monitoring" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.7.2 lbp_err lbp_get_reset_monitoring_state(int segment, int *rm_state)

Retrieves the current monitoring state (status) of a hardware reset signal monitored by LAN Bypass MCU.

Returns 0 on success or the following error codes in the event of an error:

Table 3.30 API Function "lbp_get_reset_monitoring_state" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

The reset monitoring state (rm_state) is set with bit masks defined as follows:

Table 3.31 Parameter rm_mode

Bit masks of rm_state parameter	Meaning
Bit 0 = 0b	reset monitoring state is ignored
Bit 0 = 1b	reset signal is monitored
Bit [2:1] = 00b	N/A (reserved for future use)
Bit [2:1] = 01b	Reset signal monitoring is started by BIOS
Bit [2:1] = 10b	Reset signal monitoring is started by LBPCU
Bit [2:1] = 11b	Reset monitoring timer timeout

These macros are defined in the header file and are available:

```
/* Macros for Reset Monitoring State. */
#define lbp_is_rmt_running(x)      ((x & 0x01) == 0x01) //MCU is monitoring reset signal
#define RM_ignore(x)               ((x & 0x01)==0x00) //MCU is not monitoring reset signal
#define SRM_NA(x)                  ((x & 0x06)==0x00) //Reserved for future use
#define SRM_by_BIOS(x)             ((x & 0x06)==0x02) //Monitoring is started by BIOS
#define SRM_by_LBPCU(x)            ((x & 0x06)==0x04) //Monitoring is started by LBPCU
#define SRM_by_timeout(x)          ((x & 0x06)==0x06) //Reset monitoring timer timeout
```

3.8.7.3 lbp_err lbp_set_rmt_counter(int segment, int time_100ms)

Sets the countdown value (unit: 100 ms per count) of the reset monitoring timer.

The timer will start as soon as the system powers on. When the timer expires, the LAN By-pass MCU will start to monitor the reset signal.

Returns 0 on success or the following error codes in the event of an error:

Table 3.32 API Function "lbp_set_rmt_counter" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.7.4 lbp_err lbp_get_rmt_status(int segment, int *rmt_status, int *init_time_100ms, int *current_time_100ms)

Retrieves the initial countdown value and the current countdown value of the reset monitoring timer (unit: 100 ms per count).

Returns 0 on success or the following error codes in the event of an error:

Table 3.33 API Function "lbp_get_rmt_status" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.8 LBP Message Output Level

3.8.8.1 lbp_set_message_level(int msg_level)

Sets the message output level of LBPCU:

```
msg_level:
        0 = no message output
        1 = only error message output
        2 = output error and debug message
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.34 API Function "lbp_set_message_level" Error Codes

Name	Error Code
LBP_ERR_PARAM_OUT_OF_BOUND	-109

3.8.9 Bypass Control Type

3.8.9.1 lbp_set_bypass_ctrl_type(int segment, int bp_ctrl_type)

Sets the bypass control type.

The macros are defined as:

```
/* Bypass control type. */
#define COPPER_BP_TYPE                 0xFF
#define FIBER_BP_TYPE                 0x00
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.35 API Function "lbp_set_bypass_ctrl_type" Error Codes

Name	Error Code

LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.9.2 lbp_get_bypass_ctrl_type(int segment, int *bp_ctrl_type)

Gets the bypass control type.

The macros are defined as:

```
/* Bypass control type. */
#define COPPER_BP_TYPE                  0xFF
#define FIBER_BP_TYPE                 0x00
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.36 API Function "lbp_get_bypass_ctrl_type" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.10 Advanced LAN Bypass I2C HW version

3.8.10.1 lbp_set_advanced_bypass_i2c_hw_version(int segment, int hw_version)

Sets the Advanced LAN Bypass I2C HW version.

The macros are defined as:

```
/* Advanced LAN Bypass I2C HW version */
#define LEGACY_HW 0xFF
#define NEW_HW 0x00
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.37 API Function " lbp_set_advanced_bypass_i2c_hw_version " Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.10.2 lbp_get_advanced_bypass_i2c_hw_version(int segment, int * hw_version)

Gets the Advanced LAN Bypass I2C HW version.

The macros are defined as:

```
/* Advanced LAN Bypass I2C HW version */
#define LEGACY_HW 0xFF
#define NEW_HW 0x00
```

Returns 0 on success or the following error codes in the event of an error:

Table 3.38 API Function " lbp_get_advanced_bypass_i2c_hw_version " Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.11 LBP Bypass LED

3.8.11.1 lbp_set_bypass_led(int segment, unsigned char duty, unsigned char period)

Sets the ON duty cycle and the total ON/OFF period of LED indication for a BYPASS action.

The duty settings are defined as follows:

Table 3.39 Parameter Duty

Value	Meaning
0 ~ 100	Duty of the LED

The period settings are defined as follows:

Table 3.40 Parameter Period

Value	Meaning
0 ~ 255	Period, 100 ms per unit

Returns 0 on success or the following error codes in the event of an error:

Table 3.41 API Function "lbp_set_bypass_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.11.2 lbp_get_bypass_led(int segment, unsigned char *duty, unsigned char *period)

Gets the ON duty cycle and the total ON/OFF period of LED indication for a BYPASS action.

Returns 0 on success or the following error codes in the event of an error:

Table 3.42 API Function "lbp_get_bypass_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.11.3 lbp_set_disconnect_led(int segment, unsigned char duty, unsigned char period)

Sets the ON duty cycle and the total ON/OFF period of LED indication for a DISCONNECT action.

The duty settings are defined as follows:

Table 3.43 Parameter Duty

Value	Meaning
0 ~ 100	Duty of the LED

The period settings are defined as follows:

Table 3.44 Parameter Period

Value	Meaning
0 ~ 255	Period, 100ms per unit

Returns 0 on success or the following error codes in the event of an error:

Table 3.45 API Function "lbp_set_disconnect_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.11.4 lbp_get_disconnect_led(int segment, unsigned char *duty, unsigned char *period)

Gets the ON duty cycle and the total ON/OFF period of LED indication for a DISCONNECT action.

Returns 0 on success or the following error codes in the event of an error:

Table 3.46 API Function "lbp_get_disconnect_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.12 Master/Slave Mode

Advanced LAN Bypass (I2C) do not support Master mode.

3.8.12.1 lbp_set_master_slave_mode(int segment, int master_slave_mode, int master_slave_config)

Enables or disables master/slave mode and sets the necessary parameters.

The master_slave_mode settings are defined as follows:

```
/* Mode for Master Slave Normal. */
#define NORMAL_MODE                 0x00
#define SLAVE_MODE                  0x01
#define MASTER_MODE                 0x02
```

The master_slave_config parameter is set with the bit masks defined below:

Table 3.47 Parameter master_slave_config

Parameter master_slave_config bitmasks	Meaning
Bit 0	Disable toggle pin or not 0b = Do not disable (Default) 1b = Disable toggle pin

Returns 0 on success or the following error codes in the event of an error.

Table 3.48 API Function "lbp_set_master_slave_mode" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.12.2 lbp_get_master_slave_mode(int segment, int * master_slave_mode, int * master_slave_config)

Checks the Master/Slave mode status.

Returns 0 on success or the following error codes in the event of an error:

Table 3.49 API Function "lbp_get_master_slave_mode" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.12.3 lbp_set_master_led(int segment, int action, int led_duty, int led_period_100ms)

Advanced LAN Bypass (I2C) do not support set master LED.

Sets the master LED's ON duty cycle and the total ON/OFF period of LED indication for CONNECT, DISCONNECT, and BYPASS actions.

The action settings are defined as follows:

Table 3.50 Parameter Action

Value	Meaning
1	BYPASS
2	DISCONNECT
3	CONNECT

The LED duty settings are defined as follows:

Table 3.51 Parameter Duty

Value	Meaning
0 ~ 100	Duty of the LED

The led_period settings are defined as follows:

Table 3.52 Parameter Period

Value	Meaning
0 ~ 255	Period, 100ms per unit

Returns 0 on success or the following error codes in the event of an error:

Table 3.53 API Function "lbp_set_master_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.12.4 lbp_get_master_led(int segment, int action, int * m_led_duty, int * m_led_period_100ms)

Advanced LAN Bypass (I2C) do not support get master LED.

Gets the master LED's ON duty cycle and the total ON/OFF period of LED indication for CONNECT, DISCONNECT, and BYPASS actions.

Returns 0 on success or the following error codes in the event of an error:

Table 3.54 API Function "lbp_get_master_led" Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_PARAM_NULL_POINTER	-108
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.13 Reset EEPROM

3.8.13.1 lbp_reset_eeprom(int segment)

Resets the MCU configuration EEPROM. MCU will not reset automatically. The user will need to reset the MCU by AC cycle the system. The MCU will load default settings in FW after boot up.

This command will clear MCU's EEPROM settings including Bypass control type and Advanced LAN Bypass I2C HW version. Remember to copy these two settings and set it back after AC cycle the system use 3.8.9.1 and 3.8.10.1.

Returns 0 on success or the following error codes in the event of an error:

Table 3.55 API Function " lbp_reset_eeprom " Error Codes

Name	Error Code
LBP_ERR_INTERFACE_OPEN	-119
LBP_ERR_FW_NRDY	-118
LBP_ERR_FW_CS	-117
LBP_ERR_PARAM_OUT_OF_BOUND	-109
LBP_ERR_LBP_INIT	-107
LBP_ERR_COMMAND_NOT_SUPPORT	-106

3.8.14 Scan event thread init/detach

3.8.14.1 lbp_err lbp_scan_event_init(scan_thread_parameters para)

Initializes the scan event thread (the user must configure the parameters).

The scan event thread will detect the events of all segments. If there is an event change in any segment, the thread will call the callback which the user completes the parameter, para. The callback will get two inputs: one is the segment number and the other one is the latest event. The period_100ms variable determines the sleep time of a thread between two scanning operation.

The parameter is a structure in liblbpcu.h:

```
typedef lbp_err (*callback)(int seg, int event);
typedef struct _scan_thread_parameters
{
        callback cb;//call cb when any event changed for all segments.
        int period_100ms;//0 will not start the scan thread.
}__attribute__ ((packed)) scan_thread_parameters;
```

Returns 0 on success or -1 in the event of an error.

3.8.14.2 lbp_err lbp_scan_event_detach()

Stops and detaches the scan event thread.

Returns 0 on success or -1 in the event of an error.

3.8.15 Check FW Settings

3.8.15.1 lbp_err lbp_check_FW (int segment)

Check and fix the FW settings.

Returns 0 on success.

Chapter 4 LBPCU – Advanced LAN Bypass Control Utility

Advantech provides a sample Linux command line utility (LBPCU) that uses the liblbpcu library. Basically, all functions of the liblbpcu API library can be invoked through this utility. The intended use case is for debugging/manufacturing purposes. The utility can also be used in scripts, rather than requiring full integration of the liblbpcu library into applications.

Note! *LBPCU always requires a configuration file named "lbpcu.cfg" in the same directory of binary or you could add "-config <file>" in the end of a command to specify the file name and path.*

for example: #./lbpcu -VF 1 -config /root/bin/lbpcu.cfg

In the following subsections, a lbpcu.cfg file with two segments comparable to those in Section 3.2 will be used as an example.

Note! *If the command needs input <segment>, then you could use "a" to tell LBPCU runs this command on all segments. -SMSM command doesn't support this feature.*

```
root@localhost:~# ./lbpcu -GW a
================================================
Seg 1    :
Watchdog Timer Is:                Stopped
External WDT pin status:          inactive
Toggle pin WDT mode:               active.
Toggle pin arm status:             armed
Toggle pin auto rearm:            enabled
Toggle pin status at start up:    enabled
Initial Countdown:                10.0 sec
Current Countdown:                   0.0 sec
================================================
Seg 2    :
Watchdog Timer Is:                Stopped
External WDT pin status:          active slave mode
Toggle pin WDT mode:               active.
Toggle pin arm status:             armed
Toggle pin auto rearm:            enabled
Toggle pin status at start up:    enabled
Initial Countdown:                10.0 sec
Current Countdown:                   0.0 sec
================================================
```

Note! *If the command needs input <segment>, then you could use "-d <eth_name>" to tell LBPCU runs this command on the specify eth device. This function only works with LBPCU version >=1.30, and the <eth_name> must be set in the configuration file.*

```
eth_name: eth0
SEGMENT: 1
BUS: 5e
DEV: 0
FUN: 0
```

```
root@localhost:~# ./lbpcu -VF -d eth0
Firmware version: 01.24
```

4.1 ./lbpcu

When LBPCU is started without any parameters, usage information will be displayed:

```
# ./lbpcu
Usage:   lbpcu {op}
{op}:     The operation you want to do.
-V:        Get liblbpcu version.
-VP:       Get protocol version between NIC and FW.
-VF:       Get FW version.
-VB:       Get FW bootloader version.
-n:        Get number of segment that define in config file
-GA:       Get current action for specific event on this segment
-SA:       Set current action for Event on this segment.
-SW:       Set WDT initial countdown value, 100ms/count.
-GW:       Get WDT status.
-RW:       Reset Watchdog counter to the preset value and start the countdown.
-RAW:      Reset Watchdog counter to the preset value and start the countdown for all segments.
-StopW: Stop the WDT.
-StopAW:Stop the WDT for all segments.
-ST:       Set toggle pin mode.
-SEW:      Set External watchdog mode.
-SWT:      Start wdt by toggle.
-SAWT:     Start wdt by toggle for all segments.
-WT:       Toggle the NIC to strobe/reset the WDT. Toggle pin WDT mode needs to be enabled &
armed.
-WAT:      Toggle the NIC to strobe/reset the WDT for all segments. Toggle pin WDT mode needs to be
enabled & armed.
-GLEA:     Get last event and last action on this segment.
-SCA:      Set the action on this segment. This command will change the action immediately and won't
change the event-action relationship.
-SACA:     Set the action on all segments. This command will change the action immediately and won't
```

```
change the event-action relationship.
-SRM:     Start reset monitoring by LBPCU.
-GRMS:   Get reset monitoring status.
-SRMT:   Set reset monitoring timer value.
-GRMT:   Get reset monitoring timer value.
-ML:      Set message output level
-SBCT:   Set bypass control type
-GBCT:   Get bypass control type
-SBL:     Set duty and period for LED indication of action bypass.
-GBL:     Get duty and period for LED indication of action bypass.
-SDL:     Set duty and period for LED indication of action disconnect.
-GDL:     Get duty and period for LED indication of action disconnect.
-SMSM:   Set the master slave mode for each segments.
-GMSM:   Get the master slave mode status.
-SML:     Set the duty and the period of master LED in bypass or disconnect.
-GML:     Get the duty and the period of master LED in bypass or disconnect.
-SABIHV:Set advanced bypass I2C HW version.
-GABIHV:Get advanced bypass I2C HW version.
-RE:      Reset MCU EEPROM.
-CUI:     [Upgrade] Check upgrade image
-UG:      [Upgrade] Upgrade FW
-GFM:     [Upgrade] Get FW mode
-CBTEST:evnet change callback test
-i:        Get all settings of one segment
-CheckFW: Check FW and fix eeprom settings if error in eeprom detected.
-LoadDefault: Load default segment settings.
NOTE: if the command needs input <segment>, then you could use "a" to tell lbpcu runs this command
on all segments.
        -SMSM command doesn't support this feature.
        example: ./lbpcu -GW a
NOTE: if you want to specify the config file(not using default file path or name), then you could add
"-config <file_path>".
        <file_path> means full file path which includes the file name
        example: ./lbpcu -VF 1 -config /root/test/lbpcu123.cfg
NOTE: if you want to specify the <eth_name> rather than <segment>, then you could use "-d
<eth_name>" to replace "<segment>".
        <eth_name> should be set properly in config file.
        example: ./lbpcu -VF -d eth1
NOTE: if you want to ignore the PCIE check mechnism for all segments in config file, then you could
add "-COS".
        -COS means Check One Segment only.
        example: ./lbpcu -VF 2 -COS
```

4.2 ./lbpcu -V

Displays version information for liblbpcu (See 3.8.2.1) and LBPCU:

```
# ./lbpcu -V
LBPCU version: 02.04
liblbpcu version: 02.04
LBPCU protocol version: 00.10
```

4.3 ./lbpcu -VF

Displays the LAN Bypass MCU FW version for a specific segment (See 3.8.2.3):

```
# ./lbpcu -VF
Example: Get segment_1 FW version. So use -VF 1 will get segment_1 FW version.
Command: Get FW version
Syntax: VF <segment>
Output: The FW version of this <segment>
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Retrieve the MCU FW version for Segment 1:

```
# ./lbpcu -VF 1
Firmware version: 01.24
```

4.4 ./lbpcu -VP

Gets the protocol versions of the liblbpcu and LAN Bypass MCU FW (See 3.8.2.4):

```
# ./lbpcu -VP
Example: Get segment_1 protocol version between NIC and FW. So use -VP 1 will
get segment_1 protocol version.
Command: Get protocol version
Syntax: VP <segment>
Output: Lib and FW protocol version of this <segment>
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Retrieve the protocol version of liblbpcu and the protocol version of the MCU FW on Segment

```
# ./lbpcu -VP 1
protocol version lib: 00.10; fw: 00.08
```

4.5 ./lbpcu -n

Gets the total number of segments defined in lbpcu.cfg (See section 3.7.2.2):

```
# ./lbpcu -n
Segment number in conf file : 8
Seg 1    : 03:00.0
Seg 2    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.0.255, MCU_SEGMENT: 1
Seg 3    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.0.255, MCU_SEGMENT: 2
Seg 4    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.0.255, MCU_SEGMENT: 3
Seg 5    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.0.255, MCU_SEGMENT: 4
Seg 6    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.1.255, MCU_SEGMENT: 1
Seg 7    : TYPE: BMC /dev/ipmi1 IPMI, I2C BUS: 6.1.255, MCU_SEGMENT: 2
Seg 8    : B3:00.0
```

4.6 ./lbpcu -GA

Returns the current action for the specified event on a specific segment (See 3.8.3.1):

```
# ./lbpcu –GA
Example: Current POWER_OFF event action setting is BYPASS. So use -GA 1 POWER_OFF will get BYPASS output.
Command: Get Action
Syntax: GA <segment> <event>
Output: Action defined for the <event> of this <segment>
event list:
POWER_ON
POWER_OFF
POWER_RST
WATCHDOG_START
WATCHDOG_TIMEOUT
EXTERNAL_TRIGGER
BUTTON_EVENT_1
BUTTON_EVENT_2
action list:
CONNECT
DISCONNECT
BYPASS
DO_NOT_CHANGE
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Retrieve the current action (DISCONNECT in this example) for the POWER_OFF event on Segment 1:

```
# ./lbpcu -GA 1 POWER_OFF
Event: POWER_OFF
Action: DISCONNECT
```

4.7 ./lbpcu -SA

Sets the current action for the specified event on a specific segment (See 3.8.3.2):

```
# ./lbpcu -SA
Example: Set segment_1 as BYPASS action for POWER_OFF event. So use -SA 1 POWER_OFF BYPASS will set BYPASS action for POWER_OFF event on segment_1
Command: Set Action
Syntax: SA <segment> <event> <action>
Output: N/A
event list:
POWER_ON
POWER_OFF
POWER_RST
WATCHDOG_START
WATCHDOG_TIMEOUT
EXTERNAL_TRIGGER
BUTTON_EVENT_1
BUTTON_EVENT_2
action list:
CONNECT
DISCONNECT
BYPASS
DO_NOT_CHANGE
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Set BYPASS as the current action for the POWER_OFF event on Segment 1:

```
# ./lbpcu -SA 1 POWER_OFF BYPASS
# ./lbpcu -GA 1 POWER_OFF
Event: POWER_OFF
Action: BYPASS
```

4.8 ./lbpcu -SW

Sets the initial countdown value (unit: 100 ms per count) of the WDT (See 3.8.4.1):

# ./lbpcu -SW

Example: Set segment_1 WDT initial countdown value to 100 seconds, 100ms/count. So use -SW 1 1000 will set segment_1 Watch_Dog_Timer 100 seconds.

Command: Set Watch_Dog_Timer

Syntax: SW <segment> <countdown value(100ms/count)>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109*)*

Example

Set the WDT timeout on Segment 1 to 100 s and then use the "lbpcu -GW" command to confirm the change:

```
# ./lbpcu -SW 1 1000
# ./lbpcu -GW 1
Watchdog Timer Is:                stopped
External WDT pin status:          active slave mode
Toggle pin WDT mode:              active
Toggle pin arm status:            armed
Toggle pin auto rearm:            disabled
Toggle pin status at start up:    enabled
Initial Countdown:                100.0 sec
Current Countdown:                0.0 sec
```

4.9 ./lbpcu -GW

Retrieves the current WDT status (See 3.8.4.2):

```
# ./lbpcu -GW
Example: Get segment_1 Watch_Dog_Timer status. So use -GW 1 will get
Watch_Dog_Timer status for segment_1.
Command: Get WDT status
Syntax: GW <segment>
Output: Watch_Dog_Timer status.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Retrieve the current WDT status on Segment 1:

```
# ./lbpcu -GW 1
Watchdog Timer Is:                stopped
External WDT pin status:          active slave mode
Toggle pin WDT mode:               active.
Toggle pin arm status:              armed
Toggle pin auto rearm:            disabled
Toggle pin status at start up:    enabled
Initial Countdown:                100.0 sec
Current Countdown:                  0.0 sec
```

4.10 ./lbpcu -RW

Resets the WDT counter to the initial countdown value and starts the timer (See 3.8.4.3):

```
# ./lbpcu -RW
Example: Reset segment_1 Watchdog counter to the preset value and start the countdown. So use -RW 1 will rstart countdown on segment wdt.
Command: Reset and rstart WDT
Syntax: RW <segment>
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Resets the WDT to the initial countdown value and then start the WDT on Segment 1:

```
# ./lbpcu -RW 1
```

4.11 ./lbpcu -StopW

Stops the WDT and sets the current timer value to 0 (See 3.8.4.4):

```
# ./lbpcu -StopW
Example: Stop the Watch_Dog_Timer of segment_1. So use -StopW 1 will stop WDT counting on segment_1
Command: Stop WDT
Syntax: StopW <segment>
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Stops the WDT on Segment 1:

```
# ./lbpcu -StopW 1
```

4.12 ./lbpcu -ST

Sets the toggle pin mode (See 3.8.5.1):

Advanced LAN Bypass I2C do not support this command.

```
# ./lbpcu –ST

Example: enable Toggle pin, arm Toggle pin, not auto rearm, Arm and enable by default on segment_1. So use -ST 1 ENABLE ENABLE DISABLE ENABLE to set toggle pin mode on segment_1.

Command: Set toggle pin mode

Syntax: ST <segment> <enable_pin> <arm_pin> <auto_rearm> <arm_and_enable>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Sets toggle pin mode on Segment 1 and then uses the "lbpcu -GW" command to confirm the change:

```
# ./lbpcu -ST 1 ENABLE ENABLE DISABLE ENABLE
# ./lbpcu -GW 1
Watchdog Timer Is:                stopped
External WDT pin status:          active slave mode
Toggle pin WDT mode:              active.
Toggle pin arm status:            armed
Toggle pin auto rearm:            disabled
Toggle pin status at start up:    enabled
Initial Countdown:                100.0 sec
Current Countdown:                0.0 sec
```

4.13 ./lbpcu -SEW

Sets the WDT trigger mode to global master/slave mode (See 3.8.4.5):

Advanced LAN Bypass I2C HW version LEGACY do not support MASTER_EDGE and MASTER_LEVEL.

```
# ./lbpcu -SEW
Example: Set segment_1 External watchdog mode as MASTER_EDGE. So use
-SEW 1 MASER_EDGE will set external watchdog as MASTER_EDGE mode.
Command: Set External watchdog mode
Syntax: SEW <segment> <mode>
mode:
DISABLE
SLAVE
MASTER_EDGE
MASTER_LEVEL
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Enable an external WDT with master level trigger mode and then use the "lbpcu -GW" command to read back the altered setting:

```
# ./lbpcu -SEW 1 MASTER_LEVEL
# ./lbpcu -GW 1
Watchdog Timer Is:                stopped
External WDT pin status:          active master_level mode
Toggle pin WDT mode:              active
Toggle pin arm status:            armed
Toggle pin auto rearm:            disabled
Toggle pin status at start up:    enabled
Initial Countdown:                100.0 sec
Current Countdown:                0.0 sec
```

4.14 ./lbpcu -WT

Toggle the NIC to strobe/reset the WDT. (See 0).

Advanced LAN Bypass I2C does not support this command.

```
# ./lbpcu –WT

Example: Toggle the NIC to strobe/reset the WDT on segment_1. So use -WT 1 will Toggle the NIC on segment_1.

Command: Toggle the NIC

note: Toggle pin WDT mode needs to be enabled & armed.

Syntax: WT <segment>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Strobe/reset the WDT through the toggle pin interface on Segment 1:

```
# ./lbpcu -WT 1
```

4.15 ./lbpcu -GLEA

Returns the last event and last action (i.e., the current state of bypass relays) on a specific segment (See 3.8.3.3).

```
# ./lbpcu -GLEA

Example: last event and last action on segment_1 are POWER_ON and CONNECT. So use -GLEA 1 will get POWER_ON and CONNECT output.

Command: Get last event and action

Syntax: GLEA <segment>

Output: Last event and last action.

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Gets the last event and last action on Segment 1:

```
# ./lbpcu -GLEA 1
last event: WATCHDOG_TIMEOUT
last action: DISCONNECT
```

4.16 ./lbpcu -SCA

Manually sets an action on a specific segment (See 3.8.3.4):

```
# ./lbpcu -SCA
Example: Set BYPASS action on segment_1. So use -SCA 1 BYPASS will change
segment_1's action to BYPASS immediately.
Command: Set current action
Syntax: SCA <segment> <action>
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Manually sets the BYPASS action on Segment 1 and then uses the "lbpcu -GLEA" command to check the last event and last action on Segment 1:

```
# ./lbpcu -SCA 1 BYPASS
# ./lbpcu -GLEA 1
last event: MANUAL
last action: BYPASS
```

4.17 ./lbpcu -CUI

Checks the integrity of the upgraded FW image (See 3.8.6.1):

```
# ./lbpcu –CUI
Example: Check upgrade image nmc_01_04.bin. So use -CUI nmc_01_04.bin will
check the accuracy of image file and won't change the event-action relationship.
Command: Check upgrade image file
Syntax: CUI <image file>
Output: image total line.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Checks the upgraded FW image file (in this example, lbp_fw_RU_01_24.bin):

```
# ./lbpcu -CUI lbp_fw_RU_01_24.bin
image total line is 706
```

4.18 ./lbpcu -GFM

Returns the current LAN Bypass MCU FW upgrade status on a specified segment.

This command is intended for debugging purposes during FW upgrade (See 3.8.6.2):

```
# ./lbpcu –GFM
Example: the FW mode on segment_1 is AP. So use -GFM 1 will get AP output.
Command: Get FW mode
Syntax: GFM <segment>
Output: Current FW mode.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

The following screenshots show different upgrade status of MCU FW.

Example 1

MCU FW is in application mode:

```
# ./lbpcu -GFM 1
FW is in AP
```

Example 2

MCU FW is in upgrade mode:

```
# ./lbpcu -GFM 1
FW is in bootloader
bootloader idle
```

Example 3

MCU FW upgrade process commenced:

```
# ./lbpcu -GFM 1
FW is in bootloader
bootloader write complete
```

4.19 ./lbpcu -UG

Caution:

I. DON’T use multi-process or multi-thread with -UG command. Ever since LBPCU version 1.30 it supports multi-process and multi-thread. Multi -UG command is meaningless.

II. Make sure to close all LAN bypass applications before upgrading FW to avoid unexpected situations.

III. Make sure to only use lbpcu binary >=1.26 from Advantech. (also included in library).

IV. If you use earlier utility version (< 1.26 or later), then all FW settings will be reset to Advantech manufacturing default and the bypass control type and other settings (e.g. WDT) will be required after it has been upgraded.

V. Advanced LAN Bypass (NIC) and Advanced LAN Bypass (I2C) use different images. Do not use the wrong image to upgrade.

Upgrades the LAN Bypass MCU FW on a specified segment (See 3.8.6.3):

```
# ./lbpcu -UG
Example: Upgrade FW with nmc_01_04.bin on segment_1. So use -UG 1 nmc_01_04.bin will upgrade FW to version 01_04.
Command: Upgrade FW
Syntax: UG <segment> <image file>
Output: Upgrade process.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
Upgrade the LAN Bypass MCU FW on Segment 1:
# ./lbpcu -UG 1 lbp_fw_RU_01_20.bin
FW downloading. This process will take 1 ~ 2 minutes.
FW upgrade completed.
```

4.20 ./lbpcu -SRM

Starts monitoring a reset signal on a specified segment if the reset monitoring has not been enabled yet (See 3.8.7.1).

```
# ./lbpcu -SRM
Example: Start rst monitoring by segment_1. So use -SRM 1 will start rst monitoring by segment_1.
Command: Start rst monitoring
Syntax: SRM <segment>
Output: Start rst monitoring by which segment.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SRM 1
Start rst monitoring by segment 1
```

4.21 ./lbpcu -GRMS

Retrieves the reset monitoring state on a specified segment (See 3.8.7.2).

```
# ./lbpcu -GRMS
Example: Rest monitoring state of segment_1 is monitored. So use -GRMS 1 will get monitored output.
Command: Get rst monitoring state
Syntax: GRMS <segment>
Output: Rest monitoring state
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GRMS 1
Reset Monitoring State: monitored by timeout
```

4.22 ./lbpcu -SRMT

Sets the countdown value (unit: 100ms per count) of the reset monitoring on a specified segment (See 3.8.7.3).

```
# ./lbpcu -SRMT

Example: Set segment_1 RMT initial countdown value to 100 seconds, 100ms/count. So use -SRMT 1 1000 will set segment_1 Reset_Monitoring_Timer 100seconds.

Command: Set Reset_Monitoring_Timer value

Syntax: SRMT <segment> <countdown value(100ms/count)>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SRMT 1 10
# ./lbpcu -GRMT 1
Reset Monitoring Timer Is:    Stopped
Initial Countdown:    1.0 sec
Current Countdown:    0.0 sec
```

4.23 ./lbpcu -GRMT

Retrieves the initial countdown value and current countdown value of the reset monitoring timer (unit: 100 ms per count) on a specified segment (See 3.8.7.4).

```
# ./lbpcu -GRMT

Example: Get segment_1 Reset_Monitoring_Timeout status. So use -GRMT 1 will get Reset_Monitoring_Timeout status for segment_1.

Command: Get RMT status

Syntax: GRMT <segment>

Output: Reset_Monitoring_Timer status.

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GRMT 1
Reset Monitoring Timer Is:    Stopped
Initial Countdown:    1.0 sec
Current Countdown:    0.0 sec
```

4.24 ./lbpcu -SWT

Sends the WDT start sequence through the toggle pin interface on a specified segment (See3.8.5.3).

Advanced LAN Bypass I2C does not support this command.

```
# ./lbpcu -SWT

Example: Start wdt by toggle on segment_1. So use -SWT 1 will send a spcific Toggle signal for toggling on segment_1.

Command: Start wdt by toggle

note: Toggle pin WDT mode needs to be enabled & armed.

Syntax: SWT <segment>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SWT 1
```

4.25 ./lbpcu -ML

Sets the debug message output level of LBPCU (See 3.8.8.1).

NOTE: This command can't be executed independently. This command should be added after other commands

```
# ./lbpcu –ML

Example: Output error message but debug message. Using -ML 1 will only output an error when LBPCU executes.

Command: Set message level

Syntax: ML <message level>

note:

0: no message output.

1: only error message output.

2: output error and debug message.

Output: Message output level status.

This command can't be executed independently.

This command should be added after other commands

Example: lbpcu -VF 1 -ML 2

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -VF 1 -ML 2
id: 15338086
sub id: ffff
check_conf[1]file data output:
  chip: 15338086
  bus: 3
  dev: 0
  fun: 0
conf[1].chip: 15338086
conf[1].sck: 1
conf[1].cs: 2
conf[1].si: 4
conf[1].so: 8
conf[1].seg0_gpio1en: 10000, conf[1].seg0_gpio0en: 20000
conf[1].seg0_gpio1_data: 40000, conf[1].seg0_gpio0_data: 80000
conf[1].seg0_gpio1_iodir: 400000, conf[1].seg0_gpio0_iodir: 800000
conf[1].seg0_reqen: 4, conf[1].seg0_global_wdten: 8
conf[1].seg0_req_data: 40, conf[1].seg0_global_wdt_data: 80
conf[1].seg0_req_iodir: 400, conf[1].seg0_global_wdt_iodir: 800
conf[1].CTRL_offset : 0, conf[1].CTRL_EXT_offset: 18, conf[1].FLA_offset: 1201c
check_conf[2]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 1
check_conf[3]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 2
check_conf[4]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 3
check_conf[5]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 4
check_conf[6]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 1
check_conf[7]file data output:
  type: ff
  i2c_bus: 6
  mcu_segment: 2
id: 10fb8086
sub id: 2813fe
check_conf[8]file data output:
  chip: 10fb8086
  bus: b3
  dev: 0
  fun: 0
conf[8].chip: 10fb8086
conf[8].sck: 1
conf[8].cs: 2
conf[8].si: 4
conf[8].so: 8
conf[8].seg0_gpio1en: 0, conf[8].seg0_gpio0en: 40000
conf[8].seg0_gpio1_data: 0, conf[8].seg0_gpio0_data: 4
conf[8].seg0_gpio1_iodir: 0, conf[8].seg0_gpio0_iodir: 400
conf[8].seg0_reqen: 0, conf[8].seg0_global_wdten: 0
conf[8].seg0_req_data: 0, conf[8].seg0_global_wdt_data: 0
conf[8].seg0_req_iodir: 0, conf[8].seg0_global_wdt_iodir: 0
conf[8].CTRL_offset : 0, conf[8].CTRL_EXT_offset: 0, conf[8].FLA_offset: 1001c
value 0 segment 1
pid=67800 sizeof(lbpcu_mutex_info_ds) 40
pid=67800 detect 3 bus shm created
value 0 bus 3
value 1 bus 3
value 1 segment 1
ID: 15338086 found at bus: 3 device: 0 function: 0
conf[segment].memory_bar_offset= 0x00000010
base_memory_address_l:aa200000
base_memory_address_h:00000000
fla_base_memory=80046
value 1 segment 1
send commands: 00 00
read_write_protocol checksum correct
Firmware version: 01.24
value 1 segment 1
value 0 segment 1
```

```
pid=67800 shmid 81920055
value 0 bus 3
check_destroy_shared_mutex 235 pid 67800 destroy 3 bus process shared mutex chip:
15228086
```

4.26 ./lbpcu -SBCT

Sets the bypass control type on a specified segment (See 3.8.9.1).

```
# ./lbpcu -SBCT
Example: Set bypass control type as copper. So use -SBCT COPPER will set the
bypass control type as copper.
Command: Set bypass control type.
Syntax: SBCT <segment> <bypass control type>
bypass control type:
COPPER
FIBER
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SBCT 1 FIBER
```

4.27 ./lbpcu -GBCT

Gets the bypass control type on a specified segment (See 3.8.9.2).

```
# ./lbpcu -GBCT
Example: The bypass control type is copper on segment_1. So use -GBCT 1 will get
copper bypass control type.
Command: Get bypass control type.
Syntax: GBCT <segment>
Output: Bypass control type.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GBCT 1
Bypass control type: FIBER
```

4.28 ./lbpcu -SBL

Sets the ON duty cycle and blinking period for LED indication in response to a BYPASS action on a specified segment (See 3.8.11.1).

```
# ./lbpcu –SBL
Example: Set segment 1 50(%) duty cycle of 500ms period for bypass led indication.
So use -SBL 1 50 5 will set 50(%) for duty cycle and 500ms for period.
Command: Set duty and period for bypass led indication.
Syntax: SBL <segment> <duty(%)> <period(100ms/count)>
Output: N/A
NOTE: This setting will apply to both segments on the same NMC card
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SBL 1 30 10
```

4.29 ./lbpcu -GBL

Gets the ON duty cycle and blinking period settings for LED indication in response to a BYPASS action on a specified segment (See 3.8.11.2).

```
# ./lbpcu -GBL
Example: The duty and period are 50(%) and 500ms for bypass led indication on
segment_1. So use -GBL 1 will get value of duty and period.
Command: Get duty and period for bypass led indication.
Syntax: GBL <segment>
Output: duty and period for bypass led indication.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GBL 1
Duty of bypass led: 30%
Period of bypass led: 1.0 sec
```

4.30 ./lbpcu -SDL

Sets the ON duty cycle and blinking period for LED indication in response to a DISCONNECT action on a specified segment (See 3.8.11.3).

```
# ./lbpcu -SDL
Example: Set segment 1 50(%) duty cycle of 500ms period for disconnect led indication. So use -SDL 1 50 5 will set 50(%) for duty cycle and 500ms for period.
Command: Set duty and period for disconnect led indication.
Syntax: SDL <segment> <duty(%)> <period(100ms/count)>
Output: N/A
NOTE: This setting will apply to both segments on the same NMC card
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SDL 1 30 10
```

4.31 ./lbpcu -GDL

Gets the ON duty cycle and blinking period settings for LED indication in response to a DISCONNECT action on a specified segment (See 3.8.11.4).

```
# ./lbpcu -GDL
Example: The duty and period are 50(%) and 500ms for bypass led indication on segment_1. So use -GBL 1 will get value of duty and period.
Command: Get duty and period for bypass led indication.
Syntax: GBL <segment>
Output: duty and period for bypass led indication.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GDL 1
Duty of disconnect led: 30%
Period of disconnect led: 1.0 sec
```

4.32 ./lbpcu -SMSM

Sets the segment to master mode and performs the corresponding configuration. The other segment will be set to slave mode accordingly. (See 3.8.12.1).

Advanced LAN Bypass I2C does not support MASTER mode.

```
# ./lbpcu -SMSM
Set the master slave mode and the corresponding specific config of the segment.
Syntax: SMSM <segment> <master_slave_mode> <disable_toggle_pin>
master_slave_mode:
NORMAL
MASTER
SLAVE
disable_toggle_pin:
YES
NO
Output: N/A
Example: Set the segment 1 as the master mode and disable the toggle pin
Example command: ./lbpcu -SMSM 1 MASTER YES
You can only set one segment to MASTER mode.
If you set one segment as MASTER mode, then the other segments will be set as SLAVE mode automatically.
If you set one segment as NORMAL mode, then all segments will be set as NORMAL mode automatically. Toggle pin setting is ignored.
If you set one segment as SLAVE mode, then all segments will be set as SLAVE mode automatically. Toggle pin setting is ignored.
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SMSM 1 MASTER YES
```

4.33 ./lbpcu -GMSM

Gets the master/slave mode status (See 3.8.12.2).

```
# ./lbpcu -GMSM
Get the master slave mode and the corresponding specific config of the segment.
Syntax: GMSM <segment>
Output: The master slave mode and the specific config.
Example: Get the master slave mode and the specific config of the segment 1.
```

```
Example command: ./lbpcu -GMSM 1
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GMSM 1
Master Slave Mode: MASTER_MODE
Master Slave Specific Config:
DISABLE_TOGGLE_PIN: YES
```

4.34 ./lbpcu -SML

Sets the master/slave mode LED behavior (See 3.8.12.1).

Advanced LAN Bypass I2C does not support this command.

```
# ./lbpcu -SML
Set the duty cycle and the period of the Master/Slave Mode LED to the corresponding actions.
Synatx: SML <the master segment> <action> <duty> <period (100/ms/count)>
action:
CONNECT
DISCONNECT
BYPASS
Output: N/A
Example: Set the 50 duty cycle of 1000ms (1 second) for the Master/Slave Mode LED in BYPASS action.
Example command: ./lbpcu -SML 1 BYPASS 50 10
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
root@localhost:~# ./lbpcu -SML 1 BYPASS 50 5
```

4.35 ./lbpcu -GML

Gets the master/slave mode LED behavior (See 3.8.12.4).

Advanced LAN Bypass I2C does not support this command.

```
# ./lbpcu -GML
Get the duty cycle and the period of the Master/Slave Mode LED to the corresponding actions.
Synatx: GML <the master segment> <action>
action:
CONNECT
DISCONNECT
BYPASS
Output: The duty cycle and the period of the Master/Slave Mode LED to the corresponding actions
Example: Get the duty cycle and the period of the Master/Slave Mode LED in BYPASS action.
Example command: ./lbpcu -GML 1 BYPASS
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GML 1 BYPASS
Action:BYPASS
Duty of master LED: 50%
Period of master LED: 0.5 sec
```

4.36 ./lbpcu -SABIHV

Sets the Advanced LAN Bypass I2C HW version. (See 3.8.10.1).

```
# ./lbpcu -SABIHV
Example: Set Advanced bypass I2C HW Version to Legacy HW or New HW.
Command: -SABIHV segment settings.
Syntax: -SABIHV <segment> <HW version>
HW version:
LEGACY
NEW
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SABIHV 2 NEW
```

4.37 ./lbpcu -GABIHV

Gets the Advanced LAN Bypass I2C HW version. (See 3.8.10.2).

```
# ./lbpcu -GABIHV
Example: Get Advanced bypass I2C HW Version.
Command: -GABIHV segment
Syntax: -GABIHV <segment>
Output: HW version of this <segment>
LEGACY
NEW
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -GABIHV 2
Advanced bypass I2C HW version: NEW
```

4.38 ./lbpcu -RE

Reset MCU EEPROM to 0xFF. (See 3.8.13.1).

```
# ./lbpcu -RE
Example: Reset MCU EEPROM to 0xFF.
WARNING: This command will reset MCU EEPROM, you will lose all settings
         including bypass control type and Advanced bypass I2C HW version.
         Remember to set up those two values after AC cycle the platform.
         After AC cycle. MCU will load MCU default value when it detect EEPROM
         is all 0xFF
Command: -RE segment
Syntax: -RE <segment>
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -RE 2
```

4.39 ./lbpcu -RAW

Resets the WDT counter of all segments to the initial countdown value and starts the timer (See 3.8.4.3):

```
# ./lbpcu -RAW E
Example: Reset all segments Watchdog counter to the preset value and start the countdown. So use -RAW will restart countdown on all segments wdt.
Command: Reset and restart all segments WDT
Syntax: RAW
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Resets the WDT to the initial countdown value and then starts the WDT for all segments:

```
# ./lbpcu -RAW
```

4.40 ./lbpcu -StopAW

Stops the WDT of all segments (See 3.8.4.4).

```
# ./lbpcu -StopAW E
Example: Stop the Watch_Dog_Timer of all segments. So using -StopAW will stop WDT counting on all segments
Command: Stop WDT
Syntax: StopAW
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -StopAW
```

4.41 ./lbpcu -SAWT

Sends the WDT start sequence through the toggle pin interface on all segments (See 0).

Advanced LAN Bypass I2C cannot support this command.

```
# ./lbpcu -SAWT E
Example: Start wdt by toggle on all segments. So use -SAWT will send a spcific Toggle signal for toggling on all segments.
Command: Start WDT by toggle
note: Toggle pin WDT mode needs to be enabled & armed.
Syntax: SAWT
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -SAWT
```

4.42 ./lbpcu -WAT

Toggle the NIC to strobe/reset the WDT of all segments (See 3.8.5.3).

```
# ./lbpcu -WAT E
Example: Toggle the NIC to strobe/reset the WDT on all segments. So use -WAT will Toggle the NIC on all segment.
Command: Toggle the NIC
note: Toggle pin WDT mode needs to be enabled & armed.
Syntax: WAT
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Strobe/reset the WDT through the toggle pin interface on all segments:

```
# ./lbpcu -WAT
```

4.43 ./lbpcu –SACA

Manually sets an action on all segments (See 3.8.3.4):

```
# ./lbpcu –SACA
Example: Set BYPASS action on all segments. So use -SACA BYPASS will change
all segments' action to BYPASS immediately.
Command: Set current action
Syntax: SACA <action>
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

Manually set the BYPASS action on all segments and then use the "lbpcu -GLEA" command to check the last event and last action on Segments 1 and 2:

```
# ./lbpcu -SACA BYPASS
# ./lbpcu -GLEA 1
last event: MANUAL
last action: BYPASS
# ./lbpcu -GLEA 2
last event: MANUAL
last action: BYPASS
```

4.44 ./lbpcu –CBTEST

Starts scanning an event thread and then uses a test callback function (See 3.8.14).

```
# ./lbpcu -CBTEST E
Example: Enable event change detect thread and register callback function. So use
-CBTEST to start.
Command: Enable thread and register callback function
Syntax: CBTEST
Output: N/A
Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)
```

Example

```
# ./lbpcu -RAW
# ./lbpcu -CBTEST
scan event thread id: b87a1700
seg: 1 last event: WATCHDOG_START
seg: 1 last action: BYPASS
seg: 2 last event: WATCHDOG_START
seg: 2 last action: BYPASS
seg: 1 last event: WATCHDOG_START
seg: 1 last action: BYPASS
seg: 2 last event: WATCHDOG_START
seg: 2 last action: BYPASS
seg: 1 last event: WATCHDOG_START
seg: 1 last action: BYPASS
seg: 2 last event: WATCHDOG_START
seg: 2 last action: BYPASS
seg: 1 last event: WATCHDOG_TIMEOUT
seg: 1 last action: DISCONNECT
seg 1 event changed!
enter callback seg 1 event 5
seg: 2 last event: WATCHDOG_TIMEOUT
seg: 2 last action: DISCONNECT
seg 2 event changed!
enter callback seg 2 event 5
```

4.45 ./lbpcu -i

Gets all info of a segment.

For Example

```
# ./lbpcu -i 1
Seg 1    :
------------------------------------------------
Firmware version: 01.24
Bootloader firmware version: 01.16
protocol version lib: 00.10; fw: 00.08
------------------------------------------------
Bypass control type: COPPER
------------------------------------------------
last event: POWER_ON
last action: BYPASS
------------------------------------------------
Master Slave Mode: MASTER_MODE
Master Slave Specific Config:
DISABLE_TOGGLE_PIN: YES
Action: BYPASS
Duty of master LED: 100%
Period of master LED: 1.0 sec
Action: DISCONNECT
Duty of master LED: 50%
Period of master LED: 1.0 sec
Action: CONNECT
Duty of master LED: 50%
Period of master LED: 0.1 sec
------------------------------------------------
Event: POWER_ON
Action: BYPASS
Event: POWER_OFF
Action: BYPASS
Event: POWER_RST
Action: BYPASS
Event: WATCHDOG_START
Action: CONNECT
Event: WATCHDOG_TIMEOUT
```

```
Action: BYPASS
Event: EXTERNAL_TRIGGER
Action: BYPASS
Event: BUTTON_EVENT_1
Action: BYPASS
Event: BUTTON_EVENT_2
Action: DISCONNECT
------------------------------------------------
Watchdog Timer Is:    Stopped
External WDT pin status:       inactive
Toggle pin WDT mode:       inactive.
Toggle pin arm status:    not armed
Toggle pin auto rearm:    disabled
Toggle pin status at start up:    disabled
Initial Countdown:    1.0 sec
Current Countdown:    0.0 sec
------------------------------------------------
Reset Monitoring State: monitored by timeout
Reset Monitoring Timer Is:    Stopped
Initial Countdown:    60.0 sec
Current Countdown:    0.0 sec
------------------------------------------------
Duty of bypass led: 100%
Period of bypass led: 1.0 sec
Duty of disconnect led: 50%
Period of disconnect led: 1.0 sec
```

4.46 ./lbpcu -CheckFW

Checks FW and fixes EEPROM settings if an error in EEPROM is detected.

NOTE: This command finds only invalid values in EEPROM settings and resets the settings to their default values.

```
# ./lbpcu -CheckFW
```

Example: Check segment_1 FW. So use -CheckFW 1 will check segment_1 FW and fix error.

Command: Check and fix FW.

Syntax: CheckFW <segment>

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)

Example

```
# ./lbpcu -CheckFW 1
```

4.47 ./lbpcu -LoadDefault

Resets segment settings.

```
 # ./lbpcu -LoadDefault
```

Example: Load default segment_1 settings with COPPER control type. So use -LoadDefault 1 COPPER will load default segment_1 settings.

Command: LoadDefault segment settings.

Syntax: LoadDefault <segment> <bypass control type>

bypass control type:

COPPER

FIBER

Output: N/A

Error: LBP_ERR_PARAM_OUT_OF_BOUND (-109)

Example

```
# ./lbpcu -LoadDefault 2 COPPER
WARNING: Please make sure the Advanced bypass I2C HW version is correct.
            You can use lbpcu -GABIHV to check the value.
            If the setting is wrong please use lbpcu -SABIHV to change that.
The default settings:
Seg 2   :
----------------------------------------------
Firmware version: 01.00
Bootloader firmware version: 01.00
protocol version lib: 00.10; fw: 00.10
----------------------------------------------
Bypass control type: COPPER
Advanced bypass I2C HW version: NEW
----------------------------------------------
last event: POWER_ON
last action: BYPASS
----------------------------------------------
Master Slave Mode: NORMAL_MODE
Master Slave Specific Config:
DISABLE_TOGGLE_PIN: NO
----------------------------------------------
Event: POWER_ON
Action: CONNECT
Event: POWER_OFF
Action: DISCONNECT
Event: POWER_RST
Action: CONNECT
Event: WATCHDOG_START
Action: CONNECT
Event: WATCHDOG_TIMEOUT
Action: CONNECT
Event: EXTERNAL_TRIGGER
Action: CONNECT
Event: BUTTON_EVENT_1
Action: CONNECT
Event: BUTTON_EVENT_2
Action: CONNECT
```

```
------------------------------------------------
Watchdog Timer Is:   Stopped
External WDT pin status:      inactive
Toggle pin WDT mode:      inactive.
Toggle pin arm status:   not armed
Toggle pin auto rearm:   disabled
Toggle pin status at start up:   disabled
Initial Countdown:   25.6 sec
Current Countdown:   0.0 sec
------------------------------------------------
Reset Monitoring State: monitored by timeout
Reset Monitoring Timer Is:   Stopped
Initial Countdown:   60.0 sec
Current Countdown:   0.0 sec
------------------------------------------------
Duty of bypass led: 100%
Period of bypass led: 1.0 sec
Duty of disconnect led: 50%
Period of disconnect led: 1.0 sec
```

Appendix A OS Limitation

A

Segmentation Fault - Two Linux Distributions

Please note that segmentation faults will occur when using lbpcu with Debian and Fedora because of the different default kernel configurations in these two Linux distributions.

Table A.1: Linux Distribution Compatibility

Linux distribution	Compatibility
Ubuntu	Works OK.
CentOS	Works OK.
Debian	Segmentation fault.
Fedora	Segmentation fault.

Following are two suggestions to solve this segmentation fault:

1. Rebuild the kernel and disable the config.

```
#make menuconfig
//disable kernel config
Kernel hacking ->
[ ] Filter access to /dev/mem
```

2. Modify grub.cfg to disable the config.

```
root@debian:/# vi /etc/default/grub
```

- Add "iomem=relaxed" into GRUB_CMDLINE_LINUX="".

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg
# For full documentation of the options in this file, see:
#     info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="console=ttyS0,115200 iomem=relaxed"
```

- Run update-grub.

```
root@debian:/# update-grub
```

- Reboot.

Appendix B KVM

A

PCI Passthrough

Use advanced LAN bypass in a virtual machine. The LAN bypass control is based on the BUS:DEV.FUN. Users need to use the PCI pass through feature in a virtual machine, then the LAN bypass will be dedicated to the virtual machine. Notice that each NIC/segment can only pass through to one virtual machine.

1. You need to get the original BUS:DEV.FUN.

2. Run the “virsh edit guestname” command to edit the XML configuration file for the virtual machine. Here the guestname is the name of virtual machine. You should replace it with yours.

Here is a KVM example for PCI pass through for a 4 ports advanced LAN bypass NMC card. You can add this example to pass through a 4 ports advanced LAN bypass NMC card. Notice that the green parts are the original BUS:DEV.FUN, and the red parts are the virtual machine’s BUS:DEV.FUN. You should replace the green parts with the actual values in your machine. You also need set multifunction=’on’ for function 0. The function number for virtual machine must be exactly the same as in the original machine. Otherwise the segment control for segment 1 and 2 on the same advanced LAN bypass NMC card will be abnormal.

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x19' slot='0x00' function='0x0'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x05' slot='0x01' function='0x0' multifunction='on'/>
</hostdev>
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x19' slot='0x00' function='0x1'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x05' slot='0x01' function='0x1'/>
</hostdev>
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x19' slot='0x00' function='0x2'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x05' slot='0x01' function='0x2'/>
</hostdev>
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x19' slot='0x00' function='0x3'/>
  </source>
```

```
        <address type='pci' domain='0x0000' bus='0x05' slot='0x01' function='0x3'/>
    </hostdev>
```

Reference:

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/virtualization/chap-virtualization-pci_passthrough

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-editing_a_guest_virtual_machines_configuration_file-adding_multifunction_pci_devices_to_kvm_guest_virtual_machines

Appendix C PCH I2C Bus ID changing

A

PCH I2C Bus ID changing

The PCH I2C Bus ID might change during each boot up. The I2C Bus ID sequence depends on module loading sequence during OS initialization process. Note that the bus ID might change during each boot up. There are two ways to handle this issue.

- Use apdi v1.38 or higher to generate the lbpcu.cfg during each boot up. apdi v1.38 (or higher) can detect the system I2C Bus ID and generate new lbpcu.cfg.
- Use the following sample script during each boot up which can help to modify the I801 bus ID in lbpcu.cfg.

Here is a sample script as an example:

1. Check if there is a keyword's PCH bus existing, (in our case is "I801") and get correct I801 BUS ID.
2. Read lbpcu.cfg and find TYPE is SYS_I2C. ***grep*** following line which is "BUS: AA.BB.CC"
3. For each line in 2.
   A. Get the I2C SWITCH 1 (BB) and I2C SWITCH 2 (CC)
   B. Construct correct "BUS: AA.BB.CC" line based on 1. and 3.A
   C. Use ***sed*** to modify the line.

```
# bash change_bus_number.sh

Change lbpcu.cfg's Keyword bus number.

Usage: bash change_bus_number.sh {lbpcu.cfg file} {keyword}

WARNING: you need to use "bash" instead of sh to run the script

example: #bash change_bus_number.sh ./lbpcu.cfg I801
```

Output example:

```
# bash change_bus_number.sh lbpcu.cfg I801

PCH BUS ID: 0

CORRECT_DATA: BUS: 0.4.0

CORRECT_DATA: BUS: 0.4.1

CORRECT_DATA: BUS: 0.4.2

CORRECT_DATA: BUS: 0.4.3

CORRECT_DATA: BUS: 0.6.2
```

```
CORRECT_DATA: BUS: 0.6.2

CORRECT_DATA: BUS: 0.6.2
```

Source:

```
# cat change_bus_number.sh

#!/bin/bash


if [ $# -ne 2 ];then

        echo "Change lbpcu.cfg's Keyword bus number."

        echo "Usage: bash $0 {lbpcu.cfg file} {keyword}"

        echo "WARNING: you need to use \"bash\" instead of sh to run the script"

        echo "example: #bash $0 ./lbpcu.cfg I801"

        exit 1

fi


LBPCU_FILE=$1

KEYWORD=$2

PCH_BUS_ID=`i2cdetect -l | grep -i $KEYWORD | awk '{print $1}' | awk -F "-" '{print $2}'`


#Get PCH BUS, if bus not exist, exit.

if [[ -z $PCH_BUS_ID ]];then

        echo "Error: Cannot find $KEYWORD bus on this platform, please check your environment or load proper driver"

        exit 1

fi

echo PCH BUS ID: $PCH_BUS_ID


#Which line has BUS identification.

BUS_LINE=`cat $LBPCU_FILE | grep -n "TYPE: SYS_I2C" -A 1 | grep "BUS" | awk -F "-" '{print $1}'`
```

```
#How many line need modification.

BUS_LINE_NUMBER=`cat $LBPCU_FILE | grep -n "TYPE: SYS_I2C" -A 1 | grep "BUS" | wc -l`


#Change BUS one by one

for ((i=1; i<=$BUS_LINE_NUMBER; i++))

do

        LINE=`echo $BUS_LINE | awk '{print $'$i'}'`

#        echo $i $LINE;

        LINE_DATA_SWITCH_1=`sed -n ""$LINE"p" $LBPCU_FILE | awk -F "." '{print $2}'`

        LINE_DATA_SWITCH_2=`sed -n ""$LINE"p" $LBPCU_FILE | awk -F "." '{print $3}'`

#        echo Line data: $LINE_DATA_SWITCH_1 $LINE_DATA_SWITCH_2

        #Generate correct data

        CORRECT_DATA="BUS: $PCH_BUS_ID.$LINE_DATA_SWITCH_1.$LINE_DATA_SWITCH_2"

        echo CORRECT_DATA: $CORRECT_DATA

        sed -i ""$LINE"c$CORRECT_DATA" $LBPCU_FILE

done
```

www.advantech.com

Please verify specifications before quoting. This guide is intended for reference purposes only.

All product specifications are subject to change without notice.