PHPoC

Device Programming Guide for P20

Version 1.2

Sollae Systems Co., Ltd.

PHPoC Forum: http://www.phpoc.com

Homepage: http://www.eztcp.com

Contents

1 Overview

1.1 Device

A physical device and software functions that PHPoC provides are called "device". Every device is provided as a special file form and can be used like a general file such as reading/writing files.

Structure of PHPoC file system is as follows:

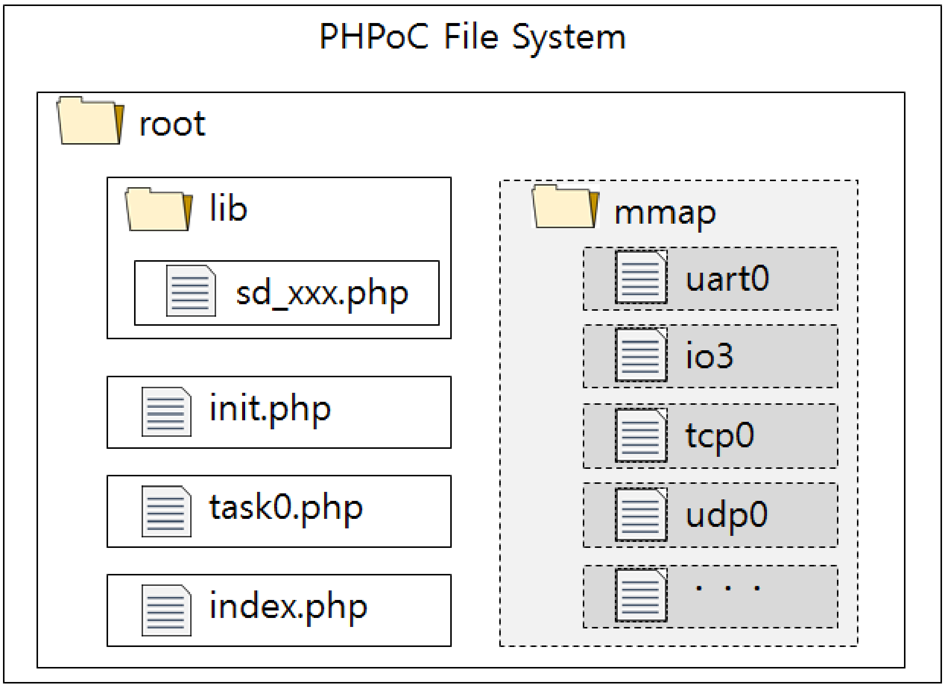


Figure 1-1 PHPoC file system

1.2 Path of Device Files

All device files of PHPoC are located in mmap (memory map) directory in root folder. To access to a specific device, you should use a path like the example below.

```
/mmap/DEVICE NAME
```

☞ ***ALL files that you upload to PHPoC are located in the root directory. You can only access to the root and /mmap directory in this file system. In addition, users are not allowed to make or remove directories.***

1.3 Types of Devices

PHPoC provides such types of devices below.

Division	Device Name
Hardware	Digital I/O(Input and Output), UART(Serial), NET(Network)
Software	TCP, UDP, ST(Software Timer)

Table 1-1 types of devices

Types may be different according to products and version of firmware.

☞ ***Refer to Appendix for more detailed device information about devices depending on the types of products.***

1.4 Steps of Using Devices

General steps of using devices are as follows:

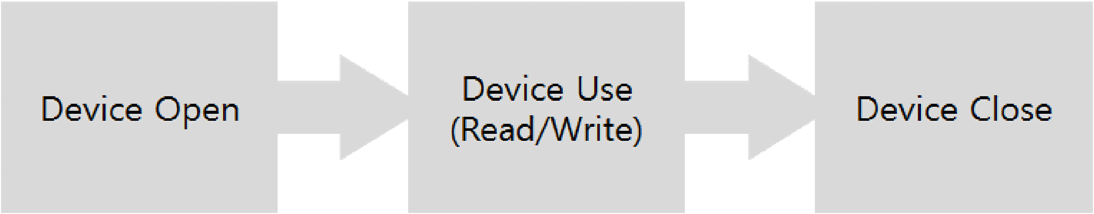


Figure 1-2 steps of using devices

1.4.1 Opening Device

pid_open function is for opening devices. This function returns pid (Peripheral ID), which is an integer value, and this value is used for accessing to devices as a unique number.

1.4.2 Using Device

After opening successfully, device, which returned pid indicates, is ready to use.

1.4.3 Closing Device

When device is not used anymore, it is needed to be closed by using pid_close function.

☞ ***Caution: It is not possible to use a physical port by new device if the port has just been used, it is not possible to be used by new device until rebooted although the device was closed. In other words, a physical port cannot be used by more than two devices before it is initialized.***

2 Digital I/O

2.1 Overview

Digital I/O can be used to monitor digital inputs or control digital outputs. This device is also used to connect LED indicators showing system status or to set types of serial communication.

- Digital I/O Structure

Every digital I/O port can have two different states which are High (or 1) and Low (or 0). Therefore, each port is matched to a binary digit as you can see below.

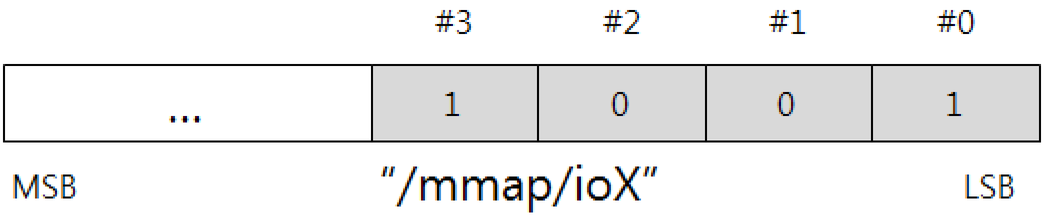


Figure 2-1 example of mapping digital I/O

2.2 Steps of Using Digital I/O

General steps of using digital I/O ports are as follows:

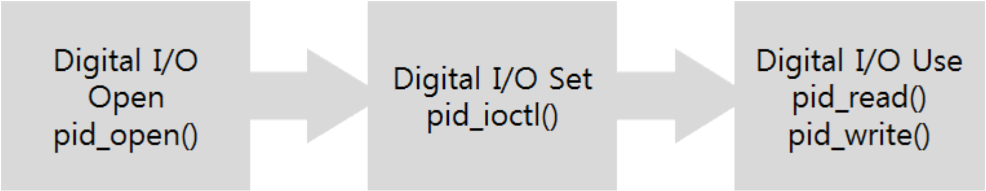

Digital I/O Open pid_open()
Digital I/O Set pid_ioctl()
Digital I/O Use pid_read() pid_write()

Figure 2-2 steps of using digital I/O

2.3 Opening Digital I/O

To open digital I/O, pid_open function is required.

```
$pid = pid_open("/mmap/io3");          // opening digital I/O 3
```

☞ ***Including LED, refer to Appendix for detailed digital I/O information depending on the types of products.***

2.4 Setting Digital I/O

Before using digital I/O, setting each port is required. To set it, set command of pid_ioctl function is used.

```
pid_ioctl($pid, "set N1[-N2] mode TYPE");
```

N1 and N2 indicate a range of multiple ports. You can only use N1 in the case of setting a single port. Available types are as follows:

2.4.1 Available Digital I/O Types

TYPE		Description
in		Input
out	-	Output
	low	Output: default LOW
	high	Output: default HIGH
led_sys		System Status LED
led_net0_act / led_net1_act		Activation of NET(net0 - wired, net1 - wireless) link LED: - successfully established network link - LOW - at the moment sending or receiving network data - HIGH
led_net0_link / led_net1_link		Network Link LED: connected to network - LOW
led_net0_rx / led_net1_rx		Network Receive LED: at the moment receiving data - LOW
led_net0_tx / led_net1_tx		Network Send LED: at the moment sending data - LOW

Table 2-1 available digital I/O types

☞ ***With products that equip LED, each port of digital I/O can be matched to physical LED. The LED is turned ON when the port state is LOW and turned OFF when it is HIGH. If you need more detailed information, please refer to circuit diagram of your product.***

- example of setting digital I/O

```
$pid = pid_open("/mmap/io3");          // open digital I/O 3
pid_ioctl($pid, "set 0 mode out");     // set port 0 to output
pid_ioctl($pid, "set 1-2 mode in");    // set port 1 ~ 2 to input
// setting port 3 to network link LED
pid_ioctl($pid, "set 3 mode led_net0_link");
// setting port 12 to network receive LED
pid_ioctl($pid, "set 12 mode led_net0_rx");
// setting port 13 to network send LED
pid_ioctl($pid, "set 13 mode led_net0_tx");
```

2.5 Using Digital I/O

2.5.1 Reading states of Digital I/O

When reading status of digital I/O ports, you can get all states of them with pid_read function or a single state with pid_ioctl function.

```
pid_read($pid, VALUE);            // read all state(in 16bits unit)
pid_ioctl($pid, "get N ITEM");    // read a single state(in a bit unit)
```

In the way of reading a single state, available ITEMs are as follows:

ITEM	Description	
mode	Return the port status in string type	I/O pin: "in", "out", "led_xxx" or etc.
		Designated to output pin of ST: "st_out"
input	Return the input port status in integer (0: LOW, 1: HIGH)	
output	Return the output port status in integer (0: LOW, 1: HIGH)	

Table 2-2 available ITMEs in reading a single state

- example of reading all digital I/O states

  The example below prints status of port 0 to 3 after setting them to input and getting the status.

```
$value = 0;
$pid = pid_open("/mmap/io3");          // open digital I/O 3
pid_ioctl($pid, "set 0-3 mode in");    // set port 0 ~ 3 to input
pid_read($pid, $value);                // read digital I/O status(16bits unit)
printf("0x%x\r\n", $value);            // output example: 0xf00f
```

- example of reading a single I/O state

  The example below prints a state of port 0 of IO number 3 after setting it to output and getting the mode and state.

```
$pid = pid_open("/mmap/io3");                 // open digital I/O 3
pid_ioctl($pid, "set 0 mode out high");       // set port 0 to output
$mode = pid_ioctl($pid, "get 0 mode");        // read a digital I/O mode
$output = pid_ioctl($pid, "get 0 output");    // read a digital I/O state
printf("%s, %d\r\n", $mode, $output);         // output example: out, 1
```

☞ ***When reading a port state with pid_ioctl function, you must use "get N input" if it is set to input port and use "get N output" if it is set to output port.***

2.5.2 Writing Values to Digital I/O

When writing values to digital I/O ports, you can set values to all of them with pid_write function or a single port with pid_ioctl function.

```
pid_write($pid, VALUE);                  // write to all ports(16 bits unit)
pid_ioctl($pid, "set N output TYPE");    // write to a single port(a bit unit)
```

- example of writing values to all ports

The following example prints all states of digital I/O ports after setting 0 ~ 7 pins of digital I/O port 3 to output port and writing a given value.

```
$value = 0;
$pid = pid_open("/mmap/io3");                  // open digital I/O 3
pid_ioctl($pid, "set 0-7 mode out");           // set port 0 ~ 7 to output
pid_read($pid, $value);                        // read status
pid_write($pid, ($value & 0xff00) | 0x0055); // write 0x55
pid_read($pid, $value);                        // read status
printf("0x%0x\r\n", $value);                   // output example: 0x0055
```

- example of writing a value to a single port

The following example prints a state of digital I/O 3's port 0 after setting it to digital output with default LOW and writing HIGH.

```
$pid = pid_open("/mmap/io3");               // open digital I/O 3
pid_ioctl($pid, "set 0 mode out low");      // set port 0 to output(LOW)
pid_ioctl($pid, "set 0 output high");       // write HIGH
$output = pid_ioctl($pid, "get 0 output");  // read state of port 0
printf("%d\r\n", $output);                  // output: 1
```

- example of setting output restriction

The following example is to compare results of when output lock to port 0 is enabled or when it is not.

```
$pid = pid_open("/mmap/io3");               // open digital I/O 3
pid_ioctl($pid, "set 0 mode out low");      // set port 0 to output(LOW)
pid_ioctl($pid, "set 0 lock");              // set port 0 to output restriction
pid_ioctl($pid, "set 0 output high");       // write HIGH to port 0
$output1 = pid_ioctl($pid, "get 0 output"); // read state of port 0
pid_ioctl($pid, "set 0 unlock");            // release the output restriction
pid_ioctl($pid, "set 0 output high");       // write HIGH to port 0 again
$output2 = pid_ioctl($pid, "get 0 output"); // read state of port 0
printf("%d, %d\r\n", $output1, $output2); // output: 0, 1
```

2.5.3 Controlling Relay

Some external products have relay ports which connected to digital output.

- example

The example below simultaneously turns all relay output ports on and off every second after turning the relay port 0 to 3 on in order.

```
$pid = pid_open("/mmap/io4");          // open digital I/O 4 (relay port)
pid_ioctl($pid, "set 7 mode out");     // Set port 7(Relay OE) to output
// Setting port 8 ~ 11 to output (relay port 0 ~ 3)
pid_ioctl($pid, "set 8-11 mode out");
pid_write($pid, 0x0100);               // turn relay port 0 on
sleep(1);
pid_write($pid, 0x0200);               // turn relay port 1 on
sleep(1);
pid_write($pid, 0x0400);               // turn relay port 2 on
sleep(1);
pid_write($pid, 0x0800);               // turn relay port 3 on
sleep(1);
pid_write($pid, 0x0f00);                // turn relay port 0 to 3 on
sleep(1);
pid_write($pid, 0x0000);               // turn relay port 0 to 3 off
```

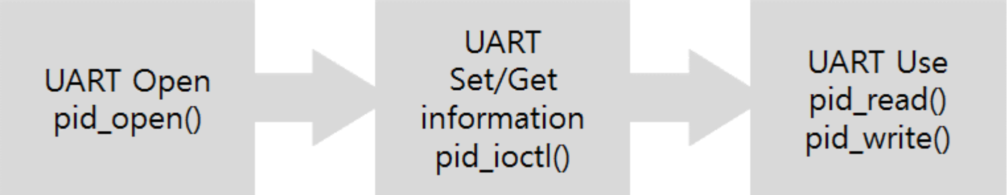
☞ ***This example is not available on products which do not have a relay port.***

3 UART

UART (Universal Asynchronous Receiver and Transmitter) is the most widely used serial communication in the world.

3.1 Steps of Using UART

General steps of using UART are as follows:



UART Open
pid_open()

UART
Set/Get
information
pid_ioctl()

UART Use
pid_read()
pid_write()

Figure 3-1 steps of using UART

3.2 Opening UART

To open UART, pid_open function is required.

```
$pid = pid_open("/mmap/uart0");                    // opening UART 0
```

☞ ***Refer to Appendix for detailed UART information depending on the types of products.***

3.3 Setting UART

Before using UART, setting parameters is required. To set them, set command of pid_ioctl function is used.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

3.3.1 Available UART Items

ITEM	VALUE	Description
baud	e.g. 9600	baud rate[bps], 2400 ~ 230400
parity	0	no parity
	1	EVEN parity
	2	ODD parity
	3	MARK parity (always 1)
	4	SPACE parity (always 0)
data	8	8 data bit
	7	7 data bit(it can be only used with parity bit)
stop	1	1 stop bit
	2	2 stop bit
flowctrl	0	no flow control
	1	RTS/CTS hardware flow control
	2	Xon/Xoff software flow control
	3	TxDE flow control for RS485

Table 3-1 available UART items

- example of setting UART

```
$pid = pid_open("/mmap/uart0");       // open UART 0
pid_ioctl($pid, "set baud 9600");     // baud rate: 9600 bps
pid_ioctl($pid, "set parity 0");      // no parity
pid_ioctl($pid, "set data 8");        // data bit length: 8
pid_ioctl($pid, "set stop 1");        // stop bit length: 1
pid_ioctl($pid, "set flowctrl 0");    // no flow control
```

3.3.2 Setting UART Communication Type

UART can be configured to RS422 or RS485 as well as RS232 depends on the types of products. For that, TxDE and setting related digital I/O are required.

This example shows how to set serial communication types of UART 0.

```
$pid = pid_open("/mmap/uart0");       // open UART 0
pid_ioctl($pid, "set baud 9600");      // baud rate: 9600 bps
pid_ioctl($pid, "set parity 0");       // parity: none
pid_ioctl($pid, "set data 8");         // data bit: 8
pid_ioctl($pid, "set stop 1");         // stop bit: 1
pid_ioctl($pid, "set flowctrl 3");     // flow control: TxDE
$pid_mode = pid_open("/mmap/io4"); // open Digital I/O 4 for UART mode
pid_ioctl($pid_mode, "set 0-3 mode out");  // set port 0~3 to output
pid_write($pid_mode, 0x05);            // RS232
//pid_write($pid_mode, 0x02);          // (remove comment sign for RS422)
//pid_write($pid_mode, 0x0c);          // (remove comment sign for RS485)
pid_close($pid_mode);
```

☞ ***These examples assumed that pin 0~3 of digital I/O number 4 are for configuration of UART types. Please check device information because the digital I/O assignment may be different according to types of products.***

☞ ***Refer to Appendix for detailed digital I/O information depending on the types of products.***

3.4 Getting Status of UART

To get various states of UART, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

3.4.1 Available UART States

ITEM	Description	Return Value	Return Type
baud	baud rate[bps]	e.g. 9600	Integer
parity	parity	0 / 1 / 2 / 3 / 4	Integer
data	data bit[bit]	8 / 7	Integer
stop	stop bit[bit]	1 / 2	Integer
flowctrl	flowctrl	0 / 1 / 2 / 3	Integer
txbuf	size of send buffer[Byte]	e.g. 1024	Integer
txfree	remaining send buffer size[Byte]	e.g. 1024	Integer
rxbuf	size of receive buffer[Byte]	e.g. 1024	Integer
rxlen	received data size[Byte]	e.g. 10	Integer

Table 3-2 available UART states

- example of getting UART states

Checking current information of UART is as follows:

```
$pid = pid_open("/mmap/uart0");                 // open UART 0
$baud = pid_ioctl($pid, "get baud");            // get baud rate
$parity = pid_ioctl($pid, "get parity");        // get parity
$data = pid_ioctl($pid, "get data");            // get data bit
$stop = pid_ioctl($pid, "get stop");            // get stop bit
$flowctrl = pid_ioctl($pid, "get flowctrl");    // get flow control mode
echo "baud = $baud\r\n";                        // output e.g.: baud = 9600
echo "parity = $parity\r\n";                    // output e.g.: parity = 0
echo "data = $data\r\n";                        // output e.g.: data = 8
echo "stop = $stop\r\n";                        // output e.g.: stop = 1
echo "flowctrl = $flowctrl\r\n";                // output e.g.: flowctrl = 0
```

3.4.2 Remaining Data Size in Send Buffer

Remaining data size in send buffer can be calculated as follows:

```
remaining data size in send buffer = size of buffer - remaining size of buffer
```

- example

This example shows how to check remaining data size in send buffer.

```
$txlen = -1;
$data = "0123456789";
$pid = pid_open("/mmap/uart0");       // open UART 0
pid_ioctl($pid, "set baud 9600");     // baud rate: 9600 bps
pid_ioctl($pid, "set parity 0");      // parity: none
pid_ioctl($pid, "set data 8");        // data bit: 8
pid_ioctl($pid, "set stop 1");        // stop bit: 1
pid_ioctl($pid, "set flowctrl 0");    // flow control: none
pid_write($pid, $data);               // write data to UART
while($txlen)
{
        $txbuf = pid_ioctl($pid, "get txbuf"); // get size of send buffer
        // get remaining size of send buffer
        $txfree = pid_ioctl($pid, "get txfree");
        $txlen = $txbuf - $txfree;    // calculate remaining data size
        echo "tx len = $txlen\r\n";   // prints the size
        usleep(1000);
}
pid_close($pid);
```

3.4.3 Received Data Size

The following shows how to get received data size of UART.

```
$rxlen = pid_ioctl($pid, "get rxlen[ $string]");
```

- Getting received data size with a string

When a string is specified after rxlen item, pid_ioctl function returns 0 until the string comes into UART. If the string comes, the function returns the whole data size including the string.

3.4.4 Remaining Size of Receive Buffer

Remaining size of receive buffer can be calculated as follows:

remaining size of receive buffer = size of buffer - received data size

- example

This example shows how to get remaining size of receive buffer.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");        // open UART 0
pid_ioctl($pid, "set baud 9600");      // baud rate: 9600 bps
pid_ioctl($pid, "set parity 0");       // parity: none
pid_ioctl($pid, "set data 8");         // data bit: 8
pid_ioctl($pid, "set stop 1");         // stop bit: 1
pid_ioctl($pid, "set flowctrl 0");     // flow control: none
$rxbuf = pid_ioctl($pid, "get rxbuf"); // get size of receive buffer
$rxlen = pid_ioctl($pid, "get rxlen"); // get received data size
$rxfree = $rxbuf - $rxlen;             // get remaining size of receive buffer
echo "rxfree = $rxfree\r\n";           // print the size
pid_close($pid);
```

3.5 Using UART

3.5.1 Reading Data

Received data from UART is stored in receive buffer. pid_read function is required to read the data.

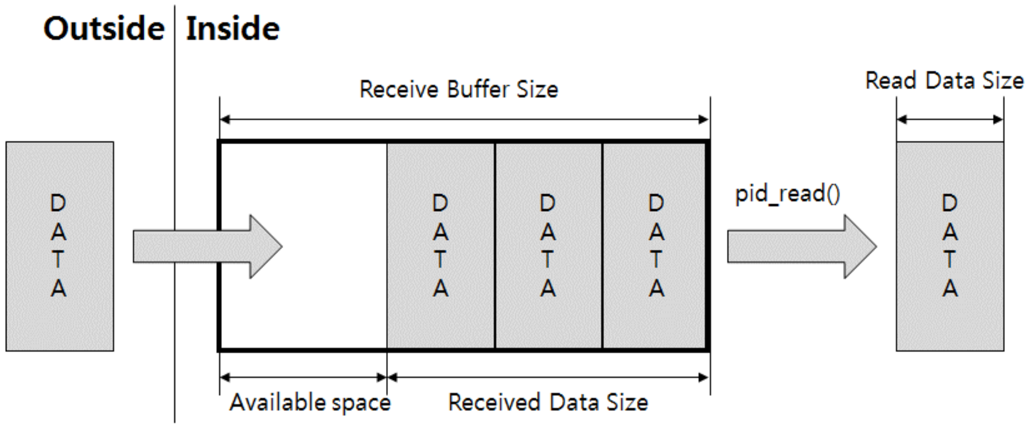


Figure 3-2 reading data from UART

The following shows how to use the pid_read function.

```
pid_read($pid, $var[, $len]);
```

Argument $var is a variable for saving the read data and $len is size of read data.

- Example

This example checks and prints received data to UART every second.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");                // open UART 0
pid_ioctl($pid, "set baud 9600");              // baud rate: 9600bps
pid_ioctl($pid, "set parity 0");               // parity: none
pid_ioctl($pid, "set data 8");                 // data bit: 8
pid_ioctl($pid, "set stop 1");                 // stop bit: 1
$rxbuf = pid_ioctl($pid, "get rxbuf");         // get size of receive buffer
while(1)
{
        $rxlen = pid_ioctl($pid, "get rxlen"); // get size of received data
        $rx_free = $rxbuf - $rxlen;            // get remaining size
        echo "$rx_free / $rxbuf\r\n";          // print remaining size
        $len = pid_read($pid, $rdata, $rxlen); // read data
        echo "len = $len / ";                  // print size of read data
        echo "rdata = $rdata\r\n";             // print read data
        sleep(1);
}
pid_close($pid);
```

3.5.2 Sending Data

Data, written by pid_write function, is stored in send buffer and transferred to outside via UART.

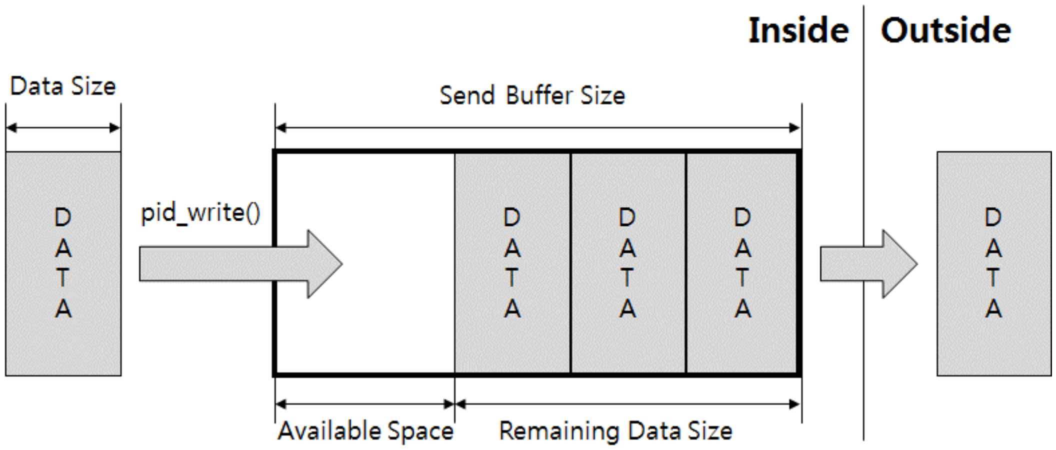


Figure 3-3 sending data to UART

The following shows how to use pid_write function.

```
pid_write($pid, $var[, $wlen]);
```

Argument $var is a variable for sending data and $wlen is a size of sending data.

- example

This example prints remaining size of send buffer and length of sent data every second.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/uart0");               // open UART 0
pid_ioctl($pid, "set baud 9600");             // baud rate: 9600bps
$txbuf = pid_ioctl($pid, "get txbuf");        // get size of send buffer
while(1)
{
        $txfree = pid_ioctl($pid, "get txfree"); // get remaining size
        echo "txfree = $txfree\r\n";             // print remaining size
        $len = pid_write($pid, $sdata, $txfree);// write data
        echo "len = $len\r\n";                   // print length of data sent
        sleep(1);
}
pid_close($pid);
```

The third argument of pid_write function means the length of writing data. The length of writing data should be less than the remaining data size of send buffer to avoid data loss. It is highly recommended to check remaining size of send buffer before sending data.

4 NET (Network)

NET means physical wired and wireless network interface.

4.1 Steps of Using NET

General steps of using NET are as follows:

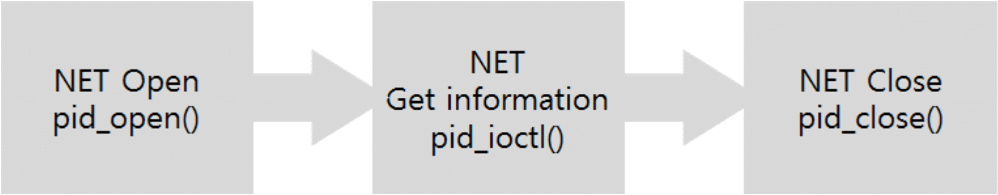


Figure 4-1 steps of using NET

4.2 Opening NET

To open NET, pid_open function is required.

```
$pid = pid_open("/mmap/net0");                    // opening NET 0
```

☞ ***Refer to Appendix for detailed NET information depending on the types of products.***

4.3 Getting Status of NET

To get a status of the NET port, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

ITEM is a name of available states.

4.3.1 Available NET States

ITEM	Description	Return Value	Return Type
hwaddr	MAC Address	e.g. 00:30:f9:00:00:01	string
ipaddr	IP Address	e.g. 10.1.0.1	string
netmask	Subnet Mask	e.g. 255.0.0.0	string
gwaddr	Gateway Address	e.g. 10.1.0.254	string
nsaddr	Name Server Address	e.g. 10.1.0.254	string
mode	10M Ethernet	10BASET	string
	100M Ethernet	100BASET	string
	WLAN Unavailable	""(an Empty String)	string
	WLAN Infrastructure	INFRA	string
	WLAN Ad-hoc	IBSS	string
	WLAN Soft AP	AP	string
speed	Ethernet Speed[Mbps]	0 / 10 / 100	integer
	WLAN Speed[100Kbps]	0 / 10 / 20 / 55 / 110 / 60 / 90 / 120 / 180 / 240 / 360 / 480 / 540	integer

Table 4-1 available NET states

- example of getting NET states

This example checks and prints various states of NET.

```
$pid = pid_open("/mmap/net0");               // open NET 0
echo pid_ioctl($pid, "get hwaddr"), "\r\n";  // get MAC address
echo pid_ioctl($pid, "get ipaddr"), "\r\n";  // get IP address
echo pid_ioctl($pid, "get netmask"), "\r\n"; // get subnet mask
echo pid_ioctl($pid, "get gwaddr"), "\r\n";  // get gateway address
echo pid_ioctl($pid, "get nsaddr"), "\r\n";  // get name server address
echo pid_ioctl($pid, "get mode"), "\r\n";    // get Ethernet mode
echo pid_ioctl($pid, "get speed"), "\r\n";   // get link speed
pid_close($pid);                             // close NET 0
```

5 TCP

TCP, in charge of transmission on TCP/IP, is one of the most fundamental protocol along with UDP, and it is widely used. This protocol contains connection phase and provides data integrity.

5.1 Steps of Using TCP

General steps of using TCP are as follows:

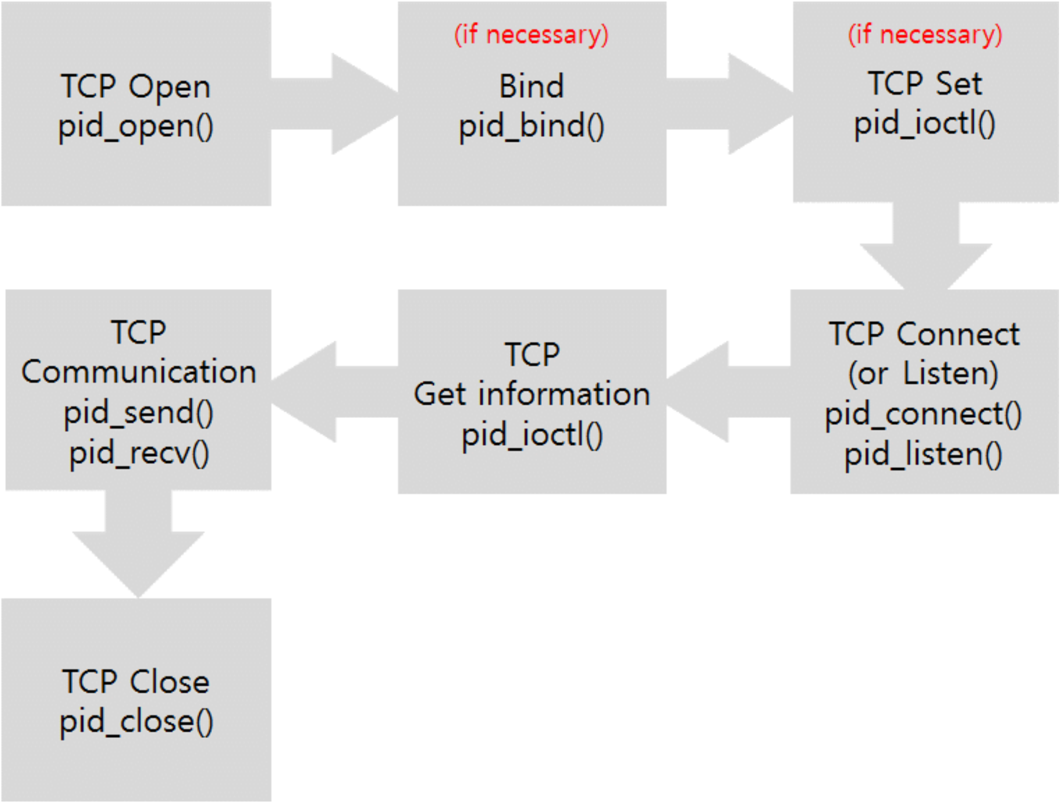


Figure 5-1 steps of using TCP

☞ ***In the case of setting a device to a TCP server, binding phase cannot be omitted.***

5.2 Opening TCP

To open TCP a session, pid_open function is required.

```
$pid = pid_open("/mmap/tcp0");                    // open TCP 0
```

☞ ***Refer to Appendix for detailed TCP information depending on the types of products.***

5.3 Setting TCP

Some parameters may be needed to set before using TCP. On SSL, SSH, TELNET or web socket communication, especially, SSL setting is required before connection by set command of pid_ioctl function.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

5.3.1 Available TCP Items

ITEM	VALUE		Description
nodelay	0		Enable Nagle algorithm
	1		Disable Nagle algorithm
api	ssl		Use SSL
	ssh		Use SSH server
	telnet		Use TELNET server
	ws		Use Web Socket server
ssl method	ssl3_client		SSL client (SSL 3.0)
	tls1_client		SSL client (TLS 1.0)
	ssl3_server		SSL server (SSL 3.0)
	tls1_server		SSL server (TLS 1.0)
ssh auth	accept		Accept SSH authorization
	reject		Reject SSH authorization
ws	path	PATH	Set path of web socket URI
	mode	0	Set data type of web socket: text
		1	Set data type of web socket: binary
	proto	PROTOCOL	Set protocol of web socket
	origin	ADDR	Specify a host to allow connection

Table 5-1 available TCP items

TCP Nagle Algorithm is to improve effective data transmission by reducing the number of segments. Thus, it may accompany a little delay.

☞ ***Note: Items by "set api" commands are only available on TCP 0 to 3. In addition, it is not possible to set another api mode once a TCP device is set to one of the modes before product reboots.***

5.3.2 How to Use SSL

PHPoC can be an SSL server or client by "set api ssl" command. The following example shows how to use it as an SSL server.

- example of an SSL server

```
$port = 1470;                                  // port number
$pid = pid_open("/mmap/tcp0");                 // open TCP 0
pid_ioctl($pid, "set api ssl");                // set api to SSL
pid_ioctl($pid, "set ssl method tls1_server"); // set SSL server mode
pid_bind($pid, "", $port);                     // binding
pid_listen($pid);                              // listen TCP connection
do
        $state = pid_ioctl($pid, "get state");
while(($state != SSL_CLOSED) && ($state != SSL_CONNECTED));

if($state == SSL_CONNECTED)
{
        echo "Connection has been established!\r\n";
        pid_close($pid);                       // close TCP connection
}
```

☞ ***It is necessary to store a certification into PHPoC before you use it as a SSL server. Create or save a certificate to your product by PHPoC Debugger.***

The following example shows how to use PHPoC as an SSL client.

- example of an SSL client

```
$addr = "10.1.0.2";                            // server's IP address
$port = 1470;                                  // server's port number
$pid = pid_open("/mmap/tcp0");                 // open TCP 0
pid_ioctl($pid, "set api ssl");                // set api to SSL
pid_ioctl($pid, "set ssl method tls1_client"); // set SSL client mode
pid_connect($pid, $addr, $port);               // connect to TCP server
do
        $state = pid_ioctl($pid, "get state");
while(($state != SSL_CLOSED) && ($state != SSL_CONNECTED));

if($state == SSL_CONNECTED)
{
        echo "Connection has been established!\r\n";
        pid_close($pid);                       // close TCP connection
}
```

☞ ***SSL communication may not be performed in case of lack of memory caused by increased memory usage of PHPoC.***

5.3.3 How to Use TELNET

PHPoC can be set as a TELNET server by using "set api telnet" command. The following is an example of a TELNET server.

- example of a TELNET server

```
$port = 23;                                     // port number
$pid = pid_open("/mmap/tcp0");                  // open TCP 0
pid_ioctl($pid, "set api telnet");              // set api to TELNET
pid_bind($pid, "", $port);                      // binding
pid_listen($pid);                               // listen TCP connection
do
        $state = pid_ioctl($pid, "get state");
while(($state != TCP_CLOSED) && ($state != TCP_CONNECTED));

if($state == TCP_CONNECTED)
{
        pid_send($pid, "Welcome to PHPoC TELNET server\r\n");
        echo "Connection has been established!\r\n";
        pid_close($pid);                        // close TCP connection
}
```

In the example above, PHPoC listens TELNET connection from clients. After connection is established, it prints a welcome message and close the connection.

5.3.4 How to Use SSH Server

PHPoC can be set to an SSH server by using "set api ssh" command. The following example shows how to use SSH server.

- example of SSH server

```
$port = 22;                                // port number
$pid = pid_open("/mmap/tcp0");             // open TCP 0
pid_ioctl($pid, "set api ssh");            // set api to SSH
pid_bind($pid, "", $port);                 // binding
pid_listen($pid);                          // listen TCP connection
while(1)
{
        $state = pid_ioctl($pid, "get state");
        if($state == SSH_AUTH)
        {
                $username = pid_ioctl($pid, "get ssh username");
                $password = pid_ioctl($pid, "get ssh password");
                echo "$username / $password\r\n";
                pid_ioctl($pid, "set ssh auth accept");
        }
        if($state == SSH_CONNECTED)
        {
                pid_send($pid, "Welcome to PHPoC SSH server\r\n");
                echo "Connection has been established!\r\n";
                pid_close($pid);
                break;
        }
}
```

In the example above, PHPoC listens TELNET connection from clients. After connection is established, it prints a username and a password from client. After that, it prints a welcome message and close the connection.

☞ ***Authentication process including user identification should be implemented in user script.***

5.3.5 How to Use Web Socket Server

PHPoC can be a web socket server by using "set api ws" command. The following example shows how to use web socket server.

- example of web socket server

This example listens TCP connection from clients. After connection is established, PHPoC prints data which is received from clients including the count of receiving data.

```
$pid = pid_open("/mmap/tcp0");            // open TCP 0
pid_ioctl($pid, "set api ws");             // set api to web socket
pid_ioctl($pid, "set ws path test");       // set URI path: /test
pid_ioctl($pid, "set ws mode 0");          // set transmission mode: text
pid_ioctl($pid, "set ws origin 10.1.0.1"); // specify a host to allow connection
pid_ioctl($pid, "set ws proto myproto");   // set protocol: myproto

pid_bind($pid, "", 0);                     // binding: default(80)

$rwbuf = "";
$count = 1;

while(1)
{
    if(pid_ioctl($pid, "get state") == TCP_CLOSED)
        pid_listen($pid);                              // listen TCP connection

    $rlen = pid_ioctl($pid, "get rxlen");
    if($rlen)
    {
        pid_recv($pid, $rwbuf);
        echo "$rwbuf\r\n";
        pid_send($pid, "echo reply $count");          // send data back
        $count++;
    }
}
pid_close($pid);
```

☞ ***You can make more powerful web interface by using the web socket with basic web server (index.php).***

☞ ***It is required to use a browser which supports web socket.***

5.4 TCP Connection

5.4.1 TCP Client (Active Connection)

Active connection means sending a TCP connection request packet to a TCP server and this host is called TCP client. To perform TCP client, pid_connect function is used.

```
pid_connect($pid, $addr, $port);
```

Argument $addr is an IP address of a TCP server and $port is a port number.

- example of TCP client

```
$pid = pid_open("/mmap/tcp0");      // open TCP
$addr = "10.1.0.2";                 // IP address of TCP server
$port = 1470;                       // TCP port
pid_connect($pid, $addr, $port);    // active TCP connection
sleep(25);
pid_close($pid);
```

5.4.2 TCP Server (Passive Connection)

Passive connection means listening a TCP connection request packet from a TCP client and this host is called TCP server. To perform a TCP server, pid_bind and pid_listen function are required.

```
pid_bind($pid, "", $port);
pid_listen($pid[, $backlog]);
```

Argument $port is a TCP port number.

- example of TCP Server

```
$pid = pid_open("/mmap/tcp0");      // open TCP
$port = 1470;                       // TCP port number
pid_bind($pid, "", $port);          // bind with the port number
pid_listen($pid);                   // passive TCP connection
sleep(25);
pid_close($pid);
```

5.5 Getting Status of TCP

To get states of TCP, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

5.5.1 Available TCP States

ITEM	Description	Return Value	Return Type
state	TCP session is closed	TCP_CLOSED	integer
	TCP session is connected	TCP_CONNECTED	integer
	TCP session waits for connection	TCP_LISTEN	integer
	SSL session is closed	SSL_CLOSED	integer
	SSL session is connected	SSL_CONNECTED	integer
	SSL session waits for connection	SSL_LISTEN	integer
	SSL session is closed	SSH_CLOSED	integer
	SSL session is connected	SSH_CONNETED	integer
	SSL session waits for connection	SSH_LISTEN	integer
	SSL authentication is completed	SSH_AUTH	integer
srcaddr	local IP address	e.g. 192.168.0.1	string
srcport	local port number	e.g. 1470	integer
dstaddr	peer IP address	e.g. 192.168.0.2	string
dstport	peer TCP number	e.g. 1470	integer
txbuf	size of send buffer[Byte]	e.g. 1152	integer
txfree	remaining send buffer size[Byte]	e.g. 1152	integer
rxbuf	size of receive buffer[Byte]	e.g. 1068	integer
rxlen	received data size[Byte]	e.g. 200	integer
ssh username	SSH user name	e.g. user	string
ssh password	SSH password	e.g. password	string

Table 5-2 available TCP states

5.5.2 TCP Session Status

Checking status of connection on TCP is very important, because TCP data communication is made after the connection phase. There are three session states: TCP_CLOSED when session is not connected, TCP_CONNECTED when session is connected and TCP_LISETN when TCP server is listening connection. SSL and SSH have also these three states: SSL_CLOSED, SSL_CONNECTED and SSL_LISTEN and SSH has an additional state about authentication (SSH_AUTH). The following shows how to get states of session.

```
$state = pid_ioctl($pid, "get state");
```

☞ ***An unknown value, which is not listed in the table above, could be returned if you try to get a state when PHPoC is connecting or closing connection. Note that it is not recommended to use these values in your script because it might be changed in the future.***

5.5.3 Remaining Data Size in Send Buffer

Remaining data size in send buffer can be calculated as follows:

```
remaining data size in send buffer = size of buffer - remaining size of buffer
```

● example

In this example, PHPoC send 8 bytes data to a server right after TCP connection is established. While sending the data, PHPoC prints remaining data size of its send buffer.

```
$tx_len = -1;
$pid = pid_open("/mmap/tcp0");                 // open TCP 0
do
{
        pid_connect($pid, "10.1.0.2", 1470);    // TCP active connection
        usleep(500000);
}
while(pid_ioctl($pid, "get state") != TCP_CONNECTED);
pid_send($pid, "01234567");                     // send 8 bytes
while($tx_len && (pid_ioctl($pid, "get state") == TCP_CONNECTED))
{
        $txbuf = pid_ioctl($pid, "get txbuf");  // get the size of send buffer
        // get the empty size of send buffer
        $txfree = pid_ioctl($pid, "get txfree");
        // calculate the size of remaining data in send buffer
        $tx_len = $txbuf - $txfree;
        echo "tx len = $tx_len\r\n";            // print the result
        usleep(10000);
}
pid_close($pid);                               // close TCP
```

5.5.4 Received Data Size

The following shows how to get received data size from TCP.

```
$rxlen = pid_ioctl($pid, "get rxlen[ $string]");
```

- Getting received data size with string

When a string is specified after rxlen item, pid_ioctl function returns 0 until the string comes. If the string comes, the function returns the whole data size including the string.

5.5.5 Remaining Size of Receive Buffer

Remaining size of receive buffer can be calculated as follows:

```
remaining size of receive buffer = size of buffer - size of received data
```

- example

This example shows how to get remaining size of receive buffer.

```
$rx_free = 1068;
$pid = pid_open("/mmap/tcp0");                 // open TCP 0
do
{
        pid_connect($pid, "10.1.0.2", 1470);   // TCP active connection
        usleep(500000);
}
while(pid_ioctl($pid, "get state") != TCP_CONNECTED);

while(($rx_free > 500) && (pid_ioctl($pid, "get state") == TCP_CONNECTED))
{
        $rxbuf = pid_ioctl($pid, "get rxbuf");   // get the size of receive buffer
        $rxlen = pid_ioctl($pid, "get rxlen");   // get the size of received data
        // calculate the available space of receive buffer
        $rx_free = $rxbuf - $rxlen;
        echo "rx free = $rx_free\r\n";           // print the result
        sleep(1);
}
pid_close($pid);                                 // close TCP
```

5.6 TCP Communication

5.6.1 Receiving TCP Data

Received data from network via TCP is stored in receive buffer. pid_recv function is required to read the data.

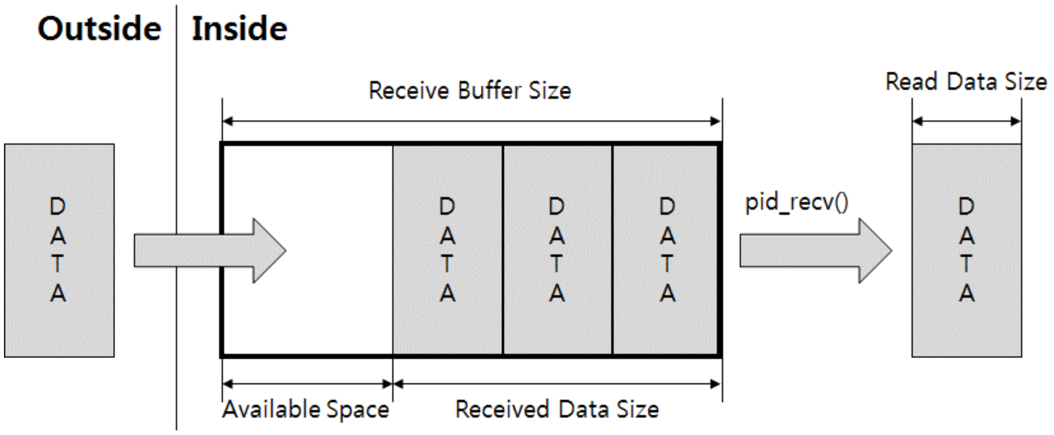


Figure 5-2 receiving TCP data

The following shows how to use pid_recv function.

```
pid_recv($pid, $value[, $len]);
```

- example

This example checks and prints received TCP data every second.

```
$rdata = "";
$pid = pid_open("/mmap/tcp0");              // open TCP 0
pid_connect($pid, "10.1.0.2", 1470);        // TCP active connection
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");   // get TCP session state
        $rxlen = pid_ioctl($pid, "get rxlen");   // get received data size
        $rlen = pid_recv($pid, $rdata, $rxlen);  // receive data
        echo "rlen = $rlen / ";                  // print received data size
        echo "rdata = $rdata\r\n";               // print received data
        if($rlen)
                $rdata = "";                     // flush receive buffer
}
while($state == TCP_CONNECTED);
pid_close($pid);
```

5.6.2 Sending TCP Data

Sent data by pid_send function is stored in send buffer and transferred to network via TCP.

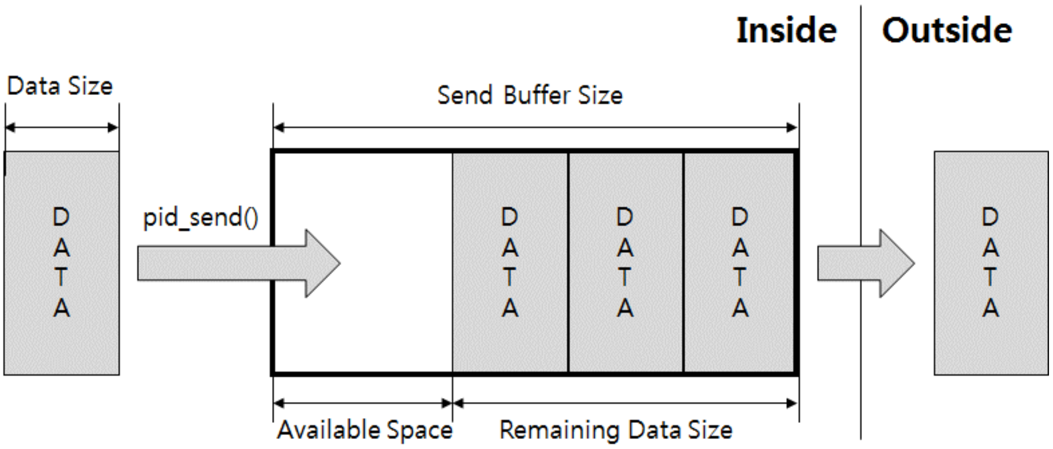


Figure 5-3 sending TCP data

The following shows how to use pid_send function.

```
pid_send($pid, $value[, $len]);
```

- example

This example sends data to network via TCP, checking available space of send buffer every second.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/tcp0");                  // open TCP 0
pid_connect($pid, "10.1.0.2", 1470);            // TCP active connection
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");  // get session state
        // get available space of send buffer
        $txfree = pid_ioctl($pid, "get txfree");
        $tx_len = pid_send($pid, $sdata, $txfree);    // send data
        echo "tx len = $tx_len\r\n";            // print size of send data
}
while($state == TCP_CONNECTED);
pid_close($pid);
```

The third argument of pid_send function means the length of sending data. The length of sending data should be less than the remaining data size of send buffer to avoid data loss. It is highly recommended to check remaining size of send buffer before sending data.

6 UDP

Although UDP does not offer data integrity and connection phase, it has good features such as simple header and prompt data transmission.

6.1 Steps of Using UDP

General steps of using UDP are as follows:

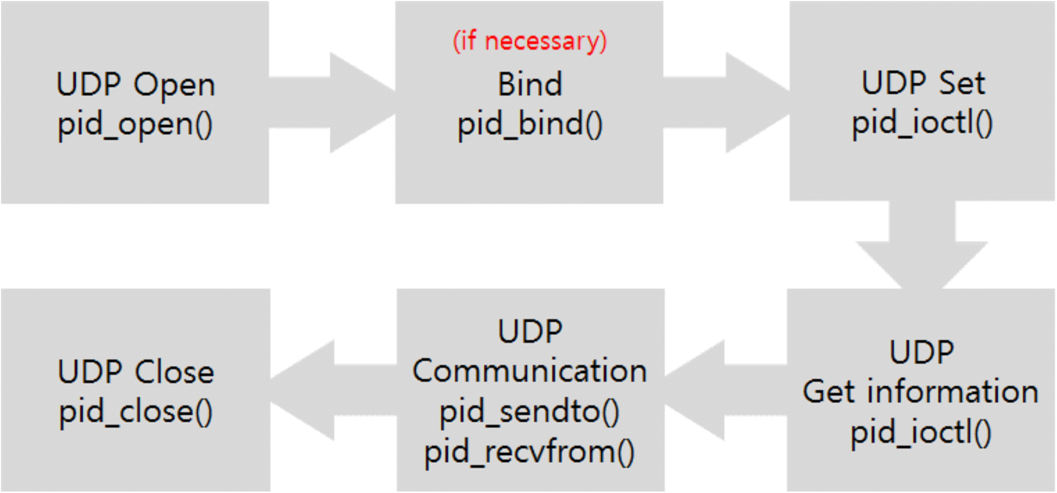


Figure 6-1 steps of using UDP

☞ ***Binding socket can be omitted if there is no requirement of prior setting or data transmission.***

6.2 Opening UDP

To open UDP, pid_open function is required.

```
$pid = pid_open("/mmap/udp0");        // open UDP 0
```

☞ ***Refer to Appendix for detailed UDP information depending on the types of products.***

6.3 Binding

Binding which uses pid_bind function is required to specify a destination IP address and to receive data from network via UDP.

```
$pid = pid_bind($pid, $addr, $port);
```

Argument $addr is an IP address and $port is port number to bind. When empty string("") is specified to the IP address, PHPoC assumes the value is the current local IP address.

☞ ***Empty string ("") value is the only option for $addr argument of function bind.***

- example of binding

```
$pid = pid_open("/mmap/udp0");      // open UDP
$port = 1470;                       // UDP port number
pid_bind($pid, "", $port);          // binding
```

6.4 Setting UDP

A destination IP address and port number can be specified before sending UDP data. This feature can omit fourth and fifth argument of pid_sendto function.

Set command of pid_ioctl function is required to set UDP.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

6.4.1 Available UDP Items

ITEM	VALUE	Description
dstaddr	e.g. 10.1.0.2	destination IP address
dstport	e.g. 1470	destination port number

Table 6-1 available UDP items

- example of setting UDP

```
$pid = pid_open("/mmap/udp0");             // open UDP 0
pid_bind($pid, "", 1470);                  // binding
pid_ioctl($pid, "set dstaddr 10.1.0.2");   // destination IP address
pid_ioctl($pid, "set dstport 1470");       // destination port number
```

6.5 Getting UDP Status

To get status of UDP, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

6.5.1 Available UDP States

ITEM	Description	Return Value	Return Type
srcaddr	source IP address	e.g. 192.168.0.1	string
srcport	source port number	e.g. 1470	integer
dstaddr	destination IP address	e.g. 192.168.0.2	string
dstport	destination port number	e.g. 1470	integer
rxlen	received data size[Byte]	e.g. 200	integer

Table 6-2 available UDP states

6.5.2 Received Data Size

To get received data size, "get rxlen" command of pid_ioctl function is required.

```
$rxlen = pid_ioctl($pid, "get rxlen");
```

- example

This example is closed after printing received data size if data comes from network while checking periodically whether there is data or not.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                  // open UDP 0
pid_bind($pid, "", 1470);                        // binding
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");   // get received data size
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen);       // receive data
                echo "$rxlen bytes\r\n";         // print size of received data
        }
        usleep(100000);
}while($rxlen == 0);                             // while receiving no data
pid_close($pid);
```

6.6 UDP Communication

6.6.1 Receiving UDP Data

To receive data from network via UDP, pid_recvfrom function is required. There are two receive buffers of UDP and the following shows how they works.

☞ ***Refer to Appendix for information about UDP receive buffer size depending on the types of products.***

- receiving UDP data from network

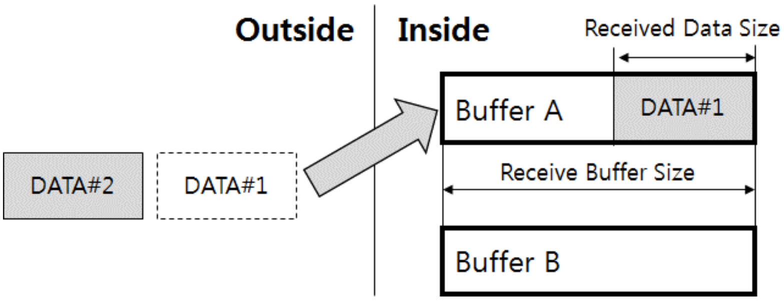


Figure 6-2 receiving UDP data from network

- reading UDP data from receive buffer

After reading data from receive buffer by calling pid_recvfrom function, PHPoC flushes the buffer.

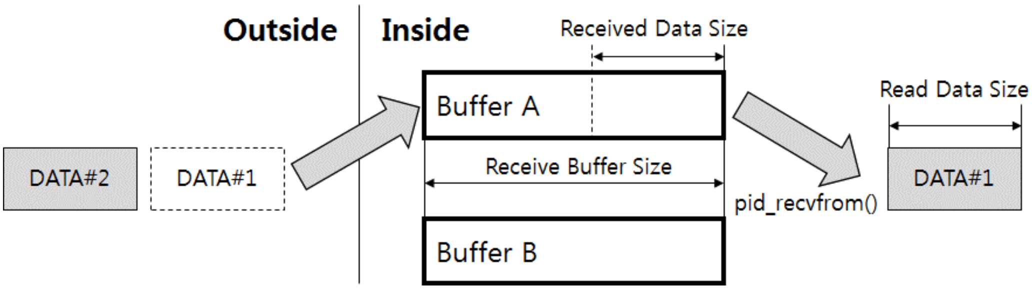


Figure 6-3 reading UDP data from receive buffer

- reading data size less than received data size

Remaining data after reading will be lost by flushing receive buffer.

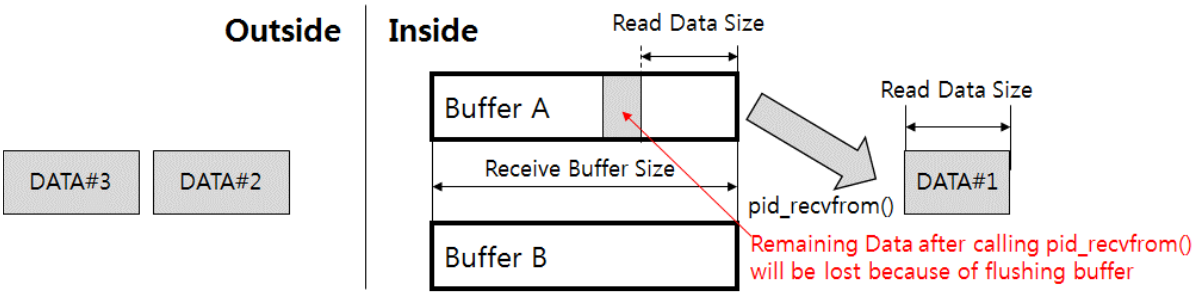


Figure 6-4 reading less size of data than received data size

- losing data by no available receive buffer

  While each of two receive buffers has data which have unread data, subsequent data from network cannot be received. Therefore, it is recommended to read data as soon as possible in received buffer right after checking received data size.

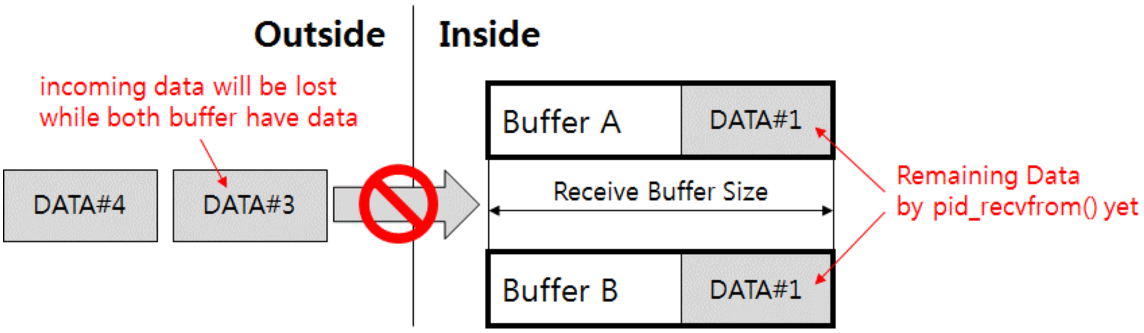


Figure 6-5 losing data by no available receive buffer

- example

This example prints received UDP data, checking if there is data comes from network every second.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                    // open UDP 0
pid_bind($pid, "", 1470);                         // binding
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");        // get received data size
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen); // receive data
                echo "$rbuf\r\n";                  // print received data
        }
        usleep(100000);
}while(1)                                         // infinite loop
```

6.6.2 Sending UDP Data

To send UDP data, pid_sendto function is required.

- example of sending UDP data

```
$sdata = "01234567";
$pid = pid_open("/mmap/udp0");                           // open UDP 0
$slen = pid_sendto($pid, $sdata, 8, 0, "10.1.0.2", 1470);  // send data
echo "slen = $slen\r\n";                              // print size of send data
pid_close($pid);
```

7 ST

PHPoC provides ST (Software Timer) device.

7.1 Steps of Using ST

General steps of using ST are as follows:

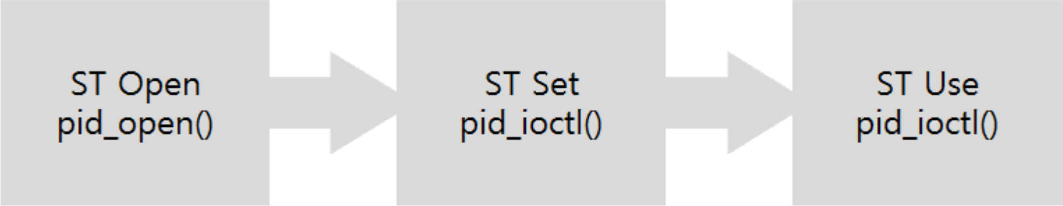


Figure 7-1 steps of using ST

7.2 Opening ST

To open ST, pid_open function is required.

```
$pid = pid_open("/mmap/st0");          // open ST 0
```

☞ ***Refer to Appendix for detailed ST information depending on the types of products.***

7.3 Setting and Using ST

To use ST, pid_ioctl function is required. There are four modes.

Mode	Description
Free mode	normal counter mode
Output Pulse mode	mode to output pulse signal through a specified pin
Output Toggle mode	mode to output toggle signal through a specified pin
Output PWM mode	mode to output infinite pulse through a specified pin

Table 7-1 ST modes

7.3.1 Common Commands

Commands listed in the table below are used in all modes of ST.

Cmd.	Sub Cmd.			Description
set	mode	free		set mode: free
		output	pulse	set mode: output Pulse
			toggle	set mode: output Toggle
			pwm	set mode: output infinite pulse
	div	sec		set unit: second
		ms		set unit: millisecond
		us		set unit: microsecond
reset	-			reset
get	state			get current state
start	-			start
stop	-			stop

Table 7-2 common commands

- Set ST Mode

ST provides both normal counter mode (free mode) and output signal mode. There are three output modes and those are pulse, toggle and pwm. The pwm is infinite pulse mode. Default value of ST mode is free mode. The following table shows how to set ST to each mode.

Mode	Syntax
free	pid_ioctl($pid, "set mode free");
pulse	pid_ioctl($pid, "set mode output pulse");
toggle	pid_ioctl($pid, "set mode output toggle");
pwm	pid_ioctl($pid, "set mode output pwm");

Table 7-3 set ST mode

- Set ST Unit

ST provides three units as follows. The default value is millisecond.

Unit	Syntax
second	pid_ioctl($pid, "set div sec");
millisecond	pid_ioctl($pid, "set div ms");
microsecond	pid_ioctl($pid, "set div us");

Table 7-4 set ST unit

- Reset

This command immediately stops operation of ST and reset.

Cmd.	Syntax
reset	pid_ioctl($pid, "reset");

Table 7-5 reset

- Get State

This command gets the current state of ST.

Cmd.	Syntax
get state	pid_ioctl($pid, "get state");

Table 7-6 get state

Return values of this command are as follows:

Return Value	Description
0	stopped
1 ~ 5	running

Table 7-7 return values of getting state

- Start

This Command starts ST.

Cmd.	Syntax
start	pid_ioctl($pid, "start");

Table 7-8 start

- Stop

This command immediately stops operation of ST. In output modes, state of output pin keeps the current state.

Cmd.	Syntax
stop	pid_ioctl($pid, "stop");

Table 7-9 stop

7.3.2 Free Mode

Free mode is normal counter mode of ST.

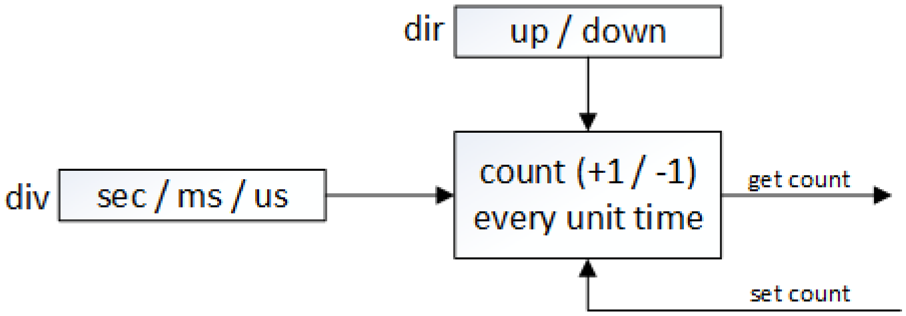


Figure 7-2 block diagram of free mode

Available commands of pid_ioctl function in free mode are as follows:

Cmd.	Sub Cmd.		Description
set	mode	free	set mode: free mode
	div	sec	set unit: second
		ms	set unit: millisecond
		us	set unit: microsecond
	dir	up	set counter direction: up counter
		down	set counter direction: down counter
	count	[T]	set default count value in down counter mode
reset	-		reset
get	count		get count value
	state		get current state
start	-		start
stop	-		stop

Table 7-10 free mode commands

- Set Counter Direction

ST can be used as both up counter and down counter. Default value of this item is up counter.

Direction	Syntax
up counter	pid_ioctl($pid, "set dir up");
down counter	pid_ioctl($pid, "set dir down");

Table 7-11 set counter direction

- Set Count

When ST is down counter mode, default value of counter can be set. How to set count value is as follows:

Cmd.	Syntax
set count	pid_ioctl($pid, "set count T");

Table 7-12 set count

Although you set a count value T, it is not applied when ST is up counter mode. Counter value starts from zero in up counter mode. Available count value in down counter mode is as follows:

Cmd.	Available Count Values
set count	0 ~ (2 to the power of 64 - 1)

Table 7-13 available count values

- Getting Count Value

Command "get count" returns a current count value.

Cmd.	Syntax
get count	pid_ioctl($pid, "get count");

Table 7-14 get count value

7.3.3 Examples of Free Mode

Command "get count" allows you to get the current count value of ST. This value represents elapsed time after timer works.

```
$tick = pid_ioctl($pid, "get count");
```

- Example of Up Counter

  This example sets ST to up counter and prints counter value in every second.

```
$pid = pid_open("/mmap/st0");                   // open ST 0
pid_ioctl($pid, "set mode free");               // set mode: free
pid_ioctl($pid, "set div sec");                 // set unit: second
pid_ioctl($pid, "set dir up");                  // set direction: up counter
pid_ioctl($pid, "start");                       // start ST
for($i=0; $i<10; $i++)
{
        $value = pid_ioctl($pid, "get count");  // read the count value
        echo "$value\r\n";                      // print the count value
        sleep(1);
}
pid_close($pid);
```

- Example of Down Counter

  This example sets ST to down counter with default count value and prints counter value in every second.

```
$pid = pid_open("/mmap/st0");                   // open ST 0
pid_ioctl($pid, "set mode free");               // set mode: free
pid_ioctl($pid, "set div sec");                 // set unit: second
pid_ioctl($pid, "set dir down");                // set direction: down counter
pid_ioctl($pid, "set count 10");                // set count value: 10
pid_ioctl($pid, "start");                       // start ST
for($i = 0; $i < 10; $i++)
{
        $value = pid_ioctl($pid, "get count");  // read the count value
        echo "$value\r\n";                      // print the count value
        sleep(1);
}
pid_close($pid);
```

7.3.4 Toggle Mode

This mode toggles the state of specified output pin of ST.

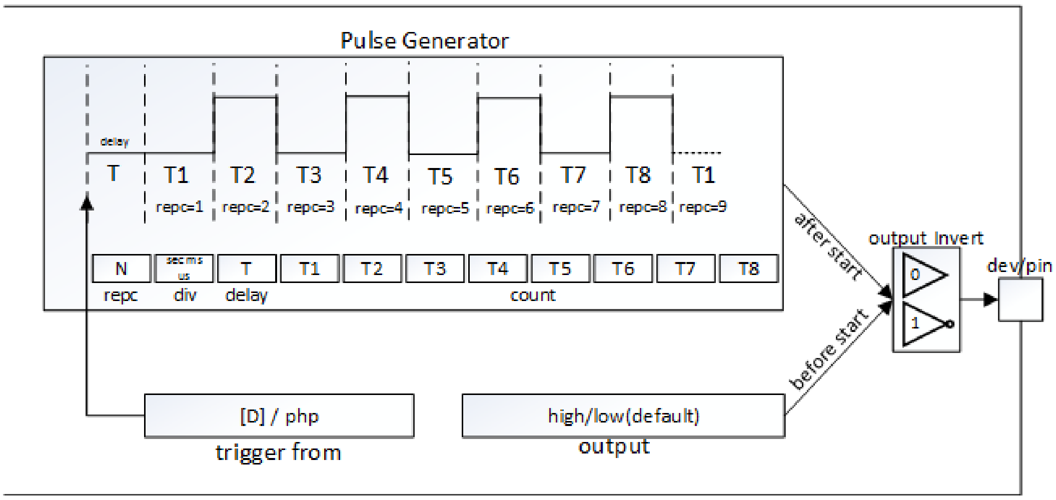


Figure 7-3 block diagram of toggle mode

Available commands in toggle mode are as follows:

Cmd.	Sub Cmd.				Description
set	mode	output	toggle		set mode: toggle
	div	sec			set unit: second
		ms			set unit: millisecond
		us			set unit: microsecond
	output	low			output LOW
		high			output HIGH
		dev	io3/io4	#pin	set output device and pin
		invert	0		not invert output
			1		invert output
	count	[T1] … [T8]			set output timing parameters
	delay	[D]			set delay before output signal
	repc	[N]			set repeat count
	trigger	from	st#		set trigger target: st0 ~ st7
			php		set trigger target: none
reset	-				reset
get	state				get state
	repc				get remaining repeat count
start	-				start
stop	-				stop

Table 7-15 toggle mode commands

- Set Output

Sub commands of "set output" command in toggle mode are as follows:

Sub Cmd.	Syntax
set output pin	pid_ioctl($pid, "set output dev io3 0"); // pin 0 of io3
output HIGH	pid_ioctl($pid, "set output high");
output LOW	pid_ioctl($pid, "set output low");

invert output	pid_ioctl($pid, "set output invert 0"); // normal output
	pid_ioctl($pid, "set output invert 1"); // inverted output

Table 7-16 set output

All commands are implemented right after each command line is executed.

- Set Delay

This command is for giving delay before PHPoC outputs signal. The unit of delay depends on the unit which is set by "set div" command.

Cmd.	Syntax
set delay	pid_ioctl($pid, "set delay D");

Table 7-17 set delay

- Set Repeat Count

This command is for setting repeat count of output. You can set any values from zero to 1 billion for the repeat count N. If you do not specify N, it is set to zero which is default value. Setting this value to zero means the maximum repeat count (1 billion).

Cmd.	Syntax
set repc	pid_ioctl($pid, "set repc N");

Table 7-18 set repeat count

- Set Count Values

This command is for defining point of time to output signal. In toggle mode, the number of count value can be minimum one and maximum eight. How to use this command is as follows:

Cmd.	Syntax
set count	pid_ioctl($pid, "set count T1 T2 ... T8");

Table 7-19 set count values

Available values for counts in toggle mode are as follows:

Unit	Available Count Values (10μs ~ half an hour)
microsecond	10 ~ 1,800,000,000
millisecond	1 ~ 1,800,000
second	1 ~ 1,800

Table 7-20 available count value

The figure below shows waveform in the case of setting just one count value T1 with delay D.

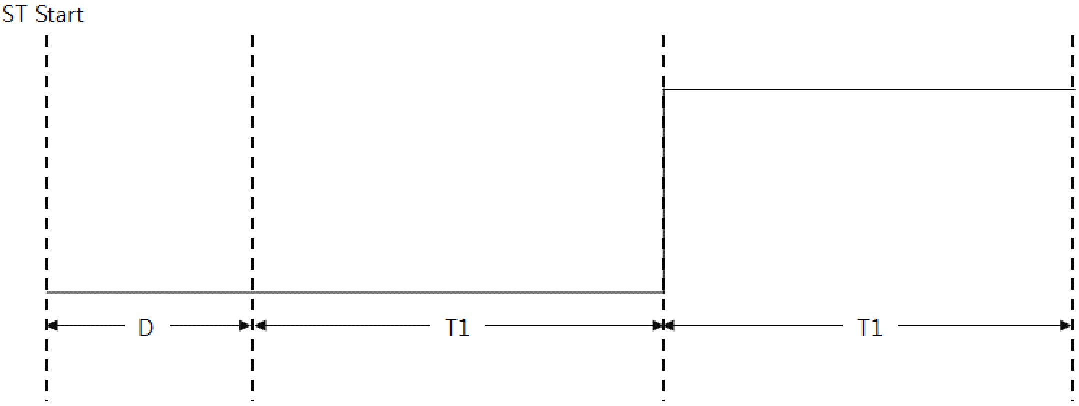


Figure 7-4 waveform of toggle mode (LOW -> HIGH)

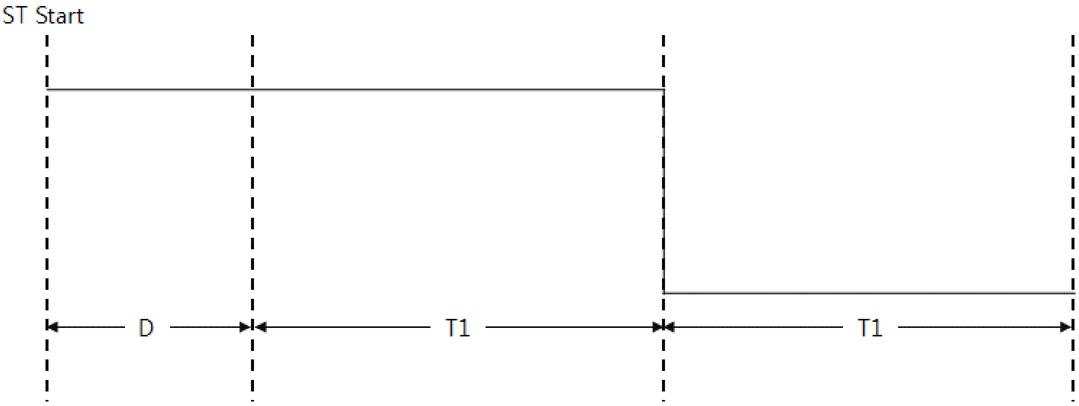


Figure 7-5 waveform of toggle mode (HIGH -> LOW)

If you set two count values or more than that, every count value is used in order. When repeat count is greater than the number of setting counts, count values are used again from the first count value. For example, waveform of setting 3 count values (T1, T2 and T3) with 4 repeat count including delay D is as follows:

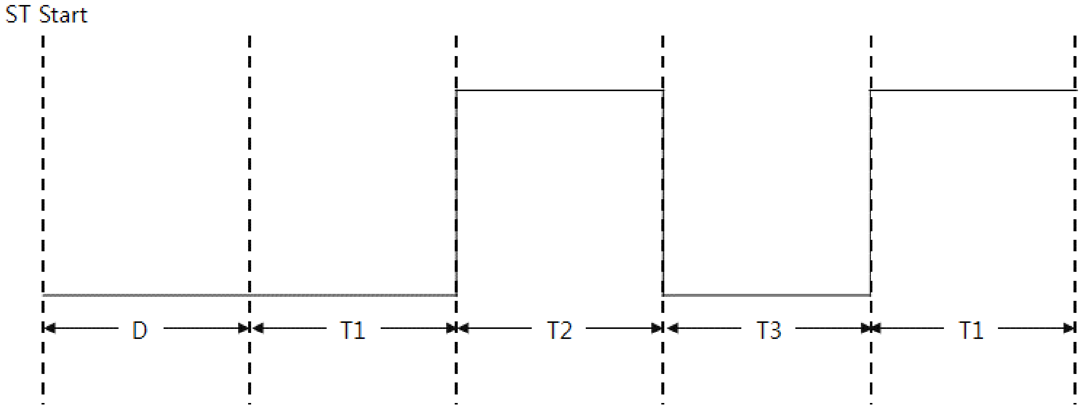


Figure 7-6 waveform of toggle mode with multiple count values

- Set Trigger

This command is used when you want to synchronize an ST start time with another ST. Target of trigger should be one of ST devices.

Target	Syntax
ST(st0/1…)	pid_ioctl($pid, "set trigger from st0");
php	pid_ioctl($pid, "set trigger from php");

Table 7-21 set trigger

Default value of trigger target is "php"(no target).

- Get Repeat Count

Command "get repc" is for reading the remaining repeat count which will be executed.

Cmd.	Syntax
get repc	pid_ioctl($pid, "get repc");

Table 7-22 get repeat count

7.3.5 Example of Toggle Mode

Toggle mode toggles output signals.

- Example of Toggle Mode

```
$pid = pid_open("/mmap/st0");                 // open ST 0
pid_ioctl($pid, "set div sec");               // set unit: second
pid_ioctl($pid, "set mode output toggle");    // set mode: toggle
pid_ioctl($pid, "set output dev io3 0");      // set output device / pin: io3 / 0
pid_ioctl($pid, "set repc 1");                // set repeat count: 1
pid_ioctl($pid, "set count 1");               // set count: T1 only
pid_ioctl($pid, "start");                     // start ST
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

The meaning of "set count" is amount of time between starting ST and output toggle signal. The figure below shows waveform of the above example.

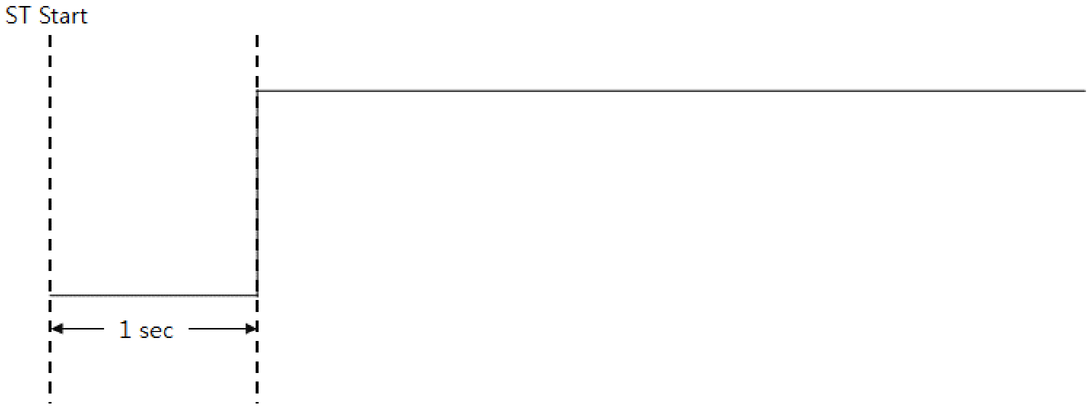


Figure 7-7 example of toggle mode

- Example of Repetitive Toggle Mode

```
$pid = pid_open("/mmap/st0");                  // open ST 0
pid_ioctl($pid, "set div sec");                // set unit: second
pid_ioctl($pid, "set mode output toggle");     // set mode: toggle
pid_ioctl($pid, "set output dev io3 0");       // set output device / pin: io3 / 0
pid_ioctl($pid, "set repc 3");                 // set repeat count: 3
pid_ioctl($pid, "set count 1 2 1");            // set count values: 1, 2 and 1
pid_ioctl($pid, "start");                      // start ST
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

In the example above, three count values (T1, T2 and T3) are set and those are 1, 2 and 1 second. The waveform is as follows:

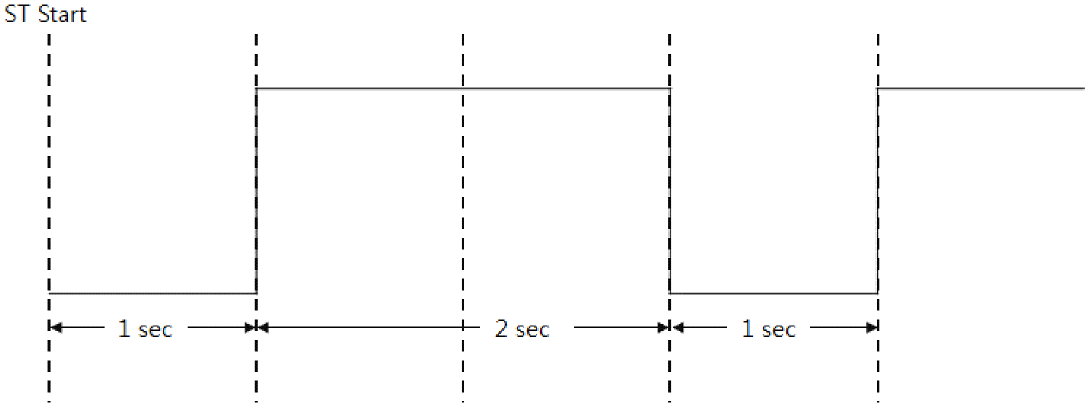


Figure 7-8 example of repetitive toggle mode

7.3.6 Pulse Mode

Pulse mode outputs square waves.

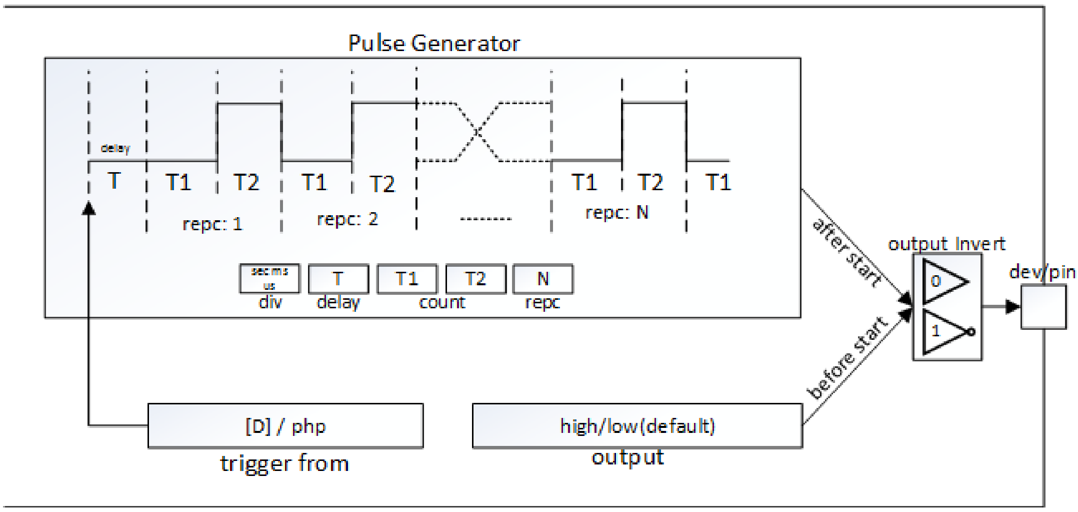


Figure 7-9 a block diagram of the pulse mode

Available commands in pulse mode are as follows:

Cmd.	Sub Cmd.				Description
set	mode	output	pulse		set mode: pulse
	div	sec			set unit: second
		ms			set unit: millisecond
		us			set unit: microsecond
	output	low			output: LOW
		high			output: HIGH
		dev	io3/4	#pin	set output device and pin
		invert	0		not invert output
			1		invert output
	count	[T1] [T2]			set output timing parameters
	delay	[D]			set delay
	repc	[N]			set repeat count
	trigger	from	st#		set trigger target: st0 ~ st7
			php		set trigger target: none
reset	-				reset
get	state				get state
	repc				get remaining repeat count
start	-				start
stop	-				stop

Table 7-23 pulse mode commands

- Set Output

Sub commands of "set output" command in pulse mode are as follows:

Sub Cmd.	Syntax
set output pin	pid_ioctl($pid, "set output dev io3 0"); // pin 0 of io3
output HIGH	pid_ioctl($pid, "set output high");
output LOW	pid_ioctl($pid, "set output low");

invert output	pid_ioctl($pid, "set output invert 1"); // normal output
	pid_ioctl($pid, "set output invert 0"); // inverted output

Table 7-24 set output

All commands are implemented right after each command line is executed.

- Set Delay

This command is for giving delay before PHPoC outputs signal. The unit of delay depends on the unit which is set by "set div" command.

Cmd.	Syntax
set delay	pid_ioctl($pid, "set delay D");

Table 7-25 setting delay

- Set Repeat Count

This command is for setting repeat count of output. You can set any values from zero to 1 billion for the repeat count N. If you do not specify N, it is set to zero which is default value. Setting this value to zero means the maximum repeat count (1 billion).

Cmd.	Syntax
set repc	pid_ioctl($pid, "set repc N");

Table 7-26 set repeat count

- Set Count Values

This command is for defining point of time to output signal. In pulse mode, two count values (T1 and T2) are required.

Cmd.	Syntax
set count	pid_ioctl($pid, "set count T1 T2");

Table 7-27 set count values

Available values for count T1 and T2 in pulse mode are as follows:

Unit	Available Count Values
microsecond	0, 10 ~ 1,800,000,000
millisecond	0 ~ 1,800,000
second	0 ~ 1,800

Table 7-28 available count values

The figure below shows waveform in the case of setting T1 and T2 with delay D in pulse mode.

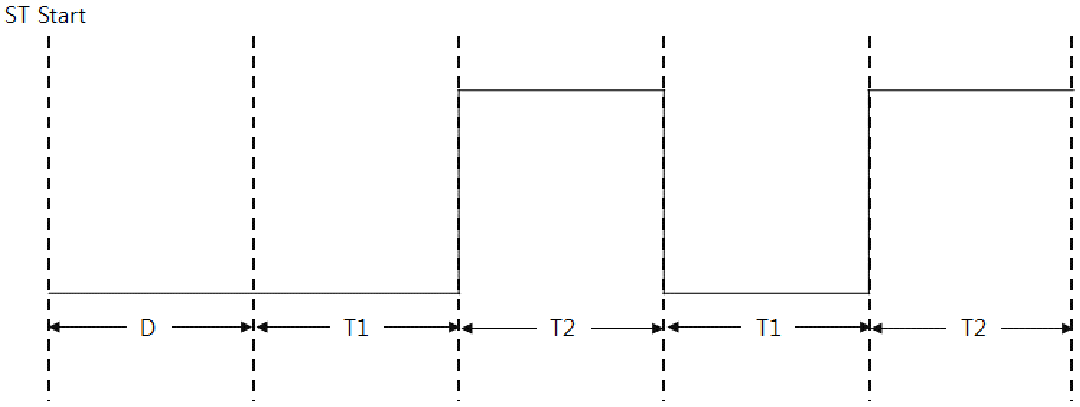


Figure 7-10 waveform of pulse mode

- Get Repeat Count

Command "get repc" is for reading the remaining repeat count which will be executed.

Cmd.	Syntax
get repc	pid_ioctl($pid, "get repc");

Table 7-29 get repeat count

7.3.7 Example of Pulse Mode

- Example of Pulse Mode (HIGH pulse)

```
$pid = pid_open("/mmap/st0");                  // open ST 0
pid_ioctl($pid, "set div sec");                // set unit: second
pid_ioctl($pid, "set mode output pulse");      // set mode: pulse
pid_ioctl($pid, "set output dev io3 0");       // set output device / pin: io3 / 0
pid_ioctl($pid, "set count 1 2");              // set count values: 1 and 2
2pid_ioctl($pid, "set repc 1");                // set repeat count: 1
pid_ioctl($pid, "start");                      // start ST
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

Pulse mode basically changes level from low to high. The timing of change depends on both division rate and count values (T1 and T2). The following figure shows waveform of the example above.

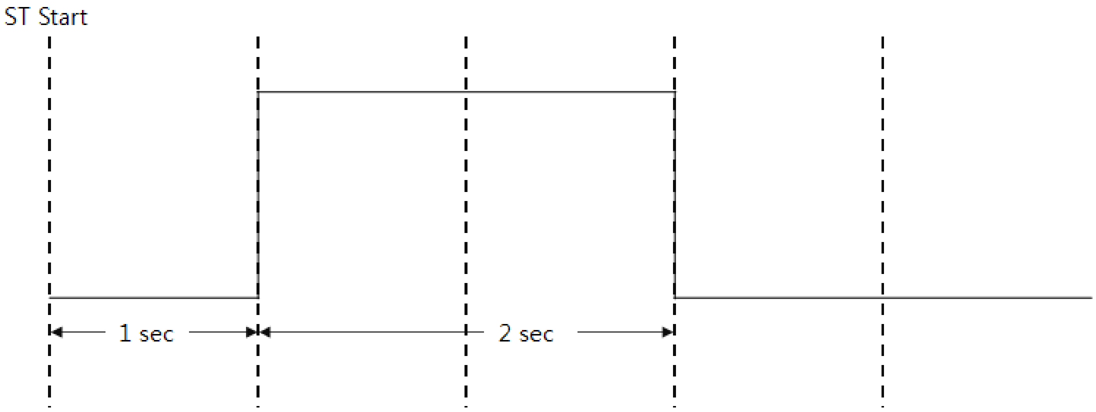


Figure 7-11 example of pulse mode (HIGH pulse)

- Example of Pulse Mode (LOW pulse)

```
$pid = pid_open("/mmap/st0");                // open ST 0
pid_ioctl($pid, "set div sec");              // set unit: second
pid_ioctl($pid, "set mode output pulse");    // set mode: pulse
pid_ioctl($pid, "set output dev io3 0");     // set output device / pin: io3 / 0
pid_ioctl($pid, "set count 1 2");            // set count values: 1 and 2
pid_ioctl($pid, "set output invert 1");      // invert output
pid_ioctl($pid, "set repc 1");               // set repeat count: 1
pid_ioctl($pid, "start");                    // start ST
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

After executing the command line "set output invert 1", all output levels are inverted including a pulse output. The figure below shows waveform of the example above.

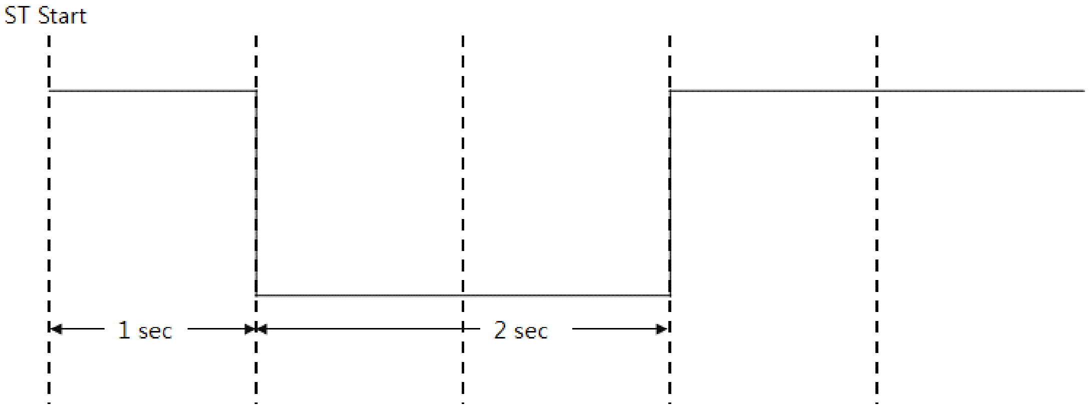


Figure 7-12 example of pulse mode (LOW pulse)

7.3.8 PWM Mode

PWM mode is called infinite pulse mode so syntax is almost the same with pulse mode but a little difference.

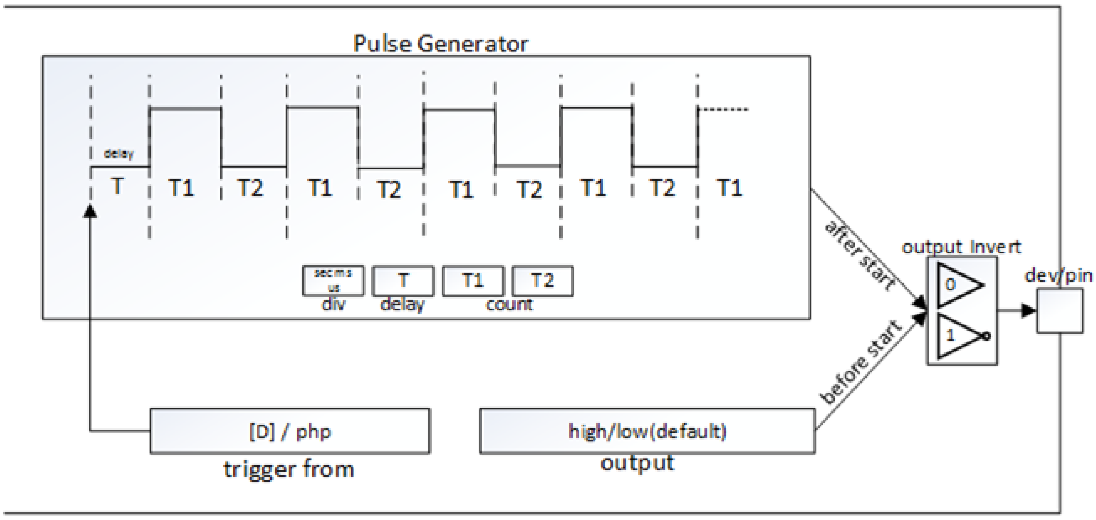


Figure 7-13 a block diagram of PWM mode

Available commands in PWM mode are as follows:

Cmd.	Sub Cmd.				Description
set	mode	output	pwm		set mode: PWM
	div	sec			set unit: second
		ms			set unit: millisecond
		us			set unit: microsecond
	output	low			output LOW
		high			output HIGH
		dev	io3/4	#pin	set output device and pin
		invert	0		not invert output
			1		invert output
	count	[T1] [T2]			set output timing parameters
	delay	[D]			set delay
	trigger	from	st#		set trigger target: st0 ~ st7
			php		set trigger target: none
reset	-				reset
get	state				get current state
start	-				start
stop	-				stop

Table 7-30 PWM mode commands

- Set Count Values

Count values defines the point of time to change levels. In PWM mode, two count values are required. How to set count values is as follows:

Cmd.	Syntax
set count	pid_ioctl($pid, "set count T1 T2");

Table 7-31 set count values

Available count values in PWM mode are as follows:

Unit	Available Count Values (0 ~ half an hour)
microsecond	0, 10 ~ 1,800,000,000
millisecond	0 ~ 1,800,000
second	0 ~ 1,800

Table 7-32 available count values

The figure below shows waveform in the case of setting T1 and T2 with delay D in PWM mode.

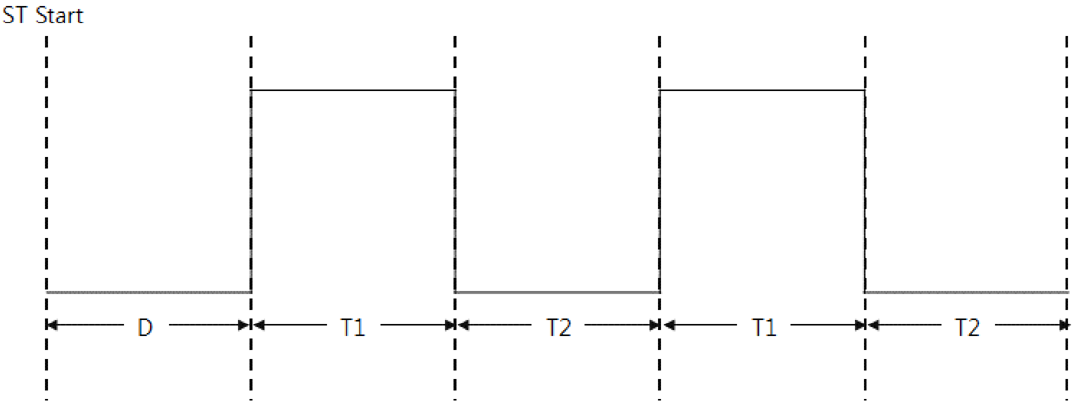


Figure 7-14 waveform of PWM mode

7.3.9 Example of PWM Mode

- Example of PWM Mode

```
$pid = pid_open("/mmap/st0");                   // open ST 0
pid_ioctl($pid, "set div sec");                 // set unit: second
pid_ioctl($pid, "set mode output pwm");  // set mode: PWM
pid_ioctl($pid, "set output dev io3 0");      // set output dev / pin: io3 / 0
pid_ioctl($pid, "set count 1 1");              // set count values 1 and 1
pid_ioctl($pid, "start");                       // start ST
sleep(10);
pid_ioctl($pid, "stop");                        // stop ST
pid_close($pid);
```

The figure below shows waveform of the example above.

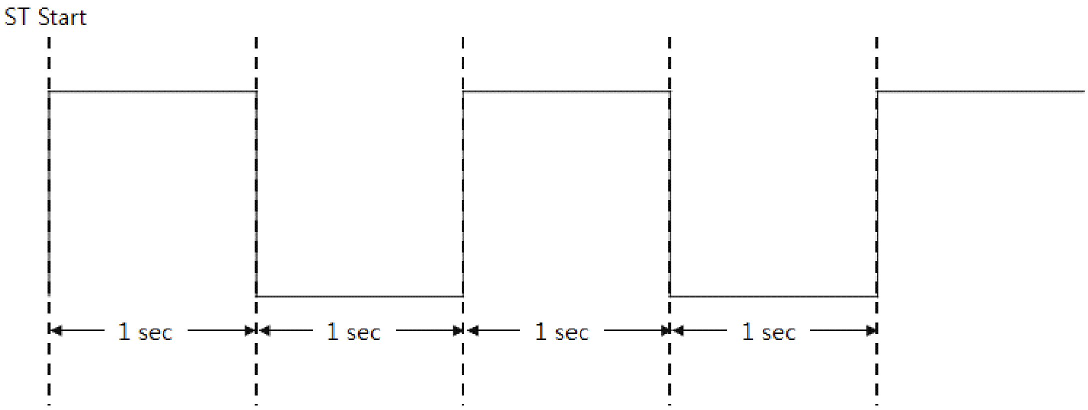


Figure 7-15 example of PWM mode

7.3.10 Trigger

Trigger command is used when you want to synchronize an ST start time with another ST. The example below shows how to synchronize ST 1 to ST 0 using trigger.

- Example of Trigger

```
$pid0 = pid_open("/mmap/st0");                // open ST 0
pid_ioctl($pid0, "set div sec");              // set unit: second
pid_ioctl($pid0, "set mode output pulse");    // set mode: pulse
pid_ioctl($pid0, "set count 1 1");            // set count values: 1 and 1
pid_ioctl($pid0, "set repc 2");               // set repeat count: 2
pid_ioctl($pid0, "set output dev io3 0");     // set output dev / pin: io3 / 0

$pid1 = pid_open("/mmap/st1");                // open ST 1
pid_ioctl($pid1, "set div sec");              // set unit: second
pid_ioctl($pid1, "set mode output pulse");    // set mode: pulse
pid_ioctl($pid1, "set trigger from st0");     // set trigger target: st0
pid_ioctl($pid1, "set count 1 1");            // set count values: 1 and 1
pid_ioctl($pid1, "set repc 2");               // set repeat count: 2
pid_ioctl($pid1, "set output dev io3 1");     // set output dev / pin: io3 / 1

pid_ioctl($pid1, "start");                    // start ST 1
pid_ioctl($pid0, "start");                    // start ST 0

while(pid_ioctl($pid1, "get state"));
pid_close($pid0);
pid_close($pid1);
```

As you see the example above, ST which you want to synchronize the output time should start before the trigger target starts.

- Error Range of ST

ST leads some error ranges and those are as follows:

Case	Error Range
Simultaneously use 2 STs	approximately 1㎲
Simultaneously use 8 STs	approximately 4㎲

Table 7-33 error ranges of ST

8 Appendix: Device Related Functions

PHPoC provides a bunch of internal functions for using devices as follows:

Function	Use Format
pid_bind	pid_bind(PID[, IP, PORT]);
pid_close	pid_close(PID);
pid_connect	pid_connect(PID, IP, PORT);
pid_ioctl	pid_ioctl(PID, COMMAND);
pid_listen	pid_listen(PID, [BACKLOG]);
pid_open	pid_open(PID[, FLAG]);
pid_read	pid_read(PID, BUF[, LEN]);
pid_recv	pid_recv(PID, BUF[, LEN, FLAG]);
pid_recvfrom	pid_recvfrom(PID, BUF[, LEN, FLAG, IP, PORT]);
pid_send	pid_send(PID, BUF[, LEN, FLAG]);
pid_sendto	pid_sendto(PID, BUF[, LEN, FLAG, IP, PORT]);
pid_write	pid_write(PID, BUF[, LEN]);

Table 8-1 device related functions

☞ ***Refer to the "PHPoC Internal Functions" document for detailed information of internal functions.***

9 Appendix: Device Information

9.1 The Number of Device Depending on Product Types

Device		PBH-101	PBH-104	PBH-204
UART		1	4	1
NET		2	2	2
TCP		5	5	5
UDP		5	5	5
I/O	Digital Input	0	0	4
	Digital Output	0	0	4
	Digital Output(LED)	8	8	8
ST		8	8	8

Table 9-1 the number of device depending on product types

9.2 Device File Path Depending on Product Types

9.2.1 UART

Product	Path
PBH-101, PBH-204	/mmap/uart0
PBH-104	/mmap/uart0
	/mmap/uart1
	/mmap/uart2
	/mmap/uart3

Table 9-2 UART

9.2.2 NET

Product	Path	Note
PBH-101, PBH-104, PBH-204	/mmap/net0	Wired LAN
	/mmap/net1	Wireless LAN

Table 9-3 NET

9.2.3 TCP

Product	Path
PBH-101, PBH-104, PBH-204	/mmap/tcp0
	/mmap/tcp1
	/mmap/tcp2
	/mmap/tcp3
	/mmap/tcp4

Table 9-4 TCP

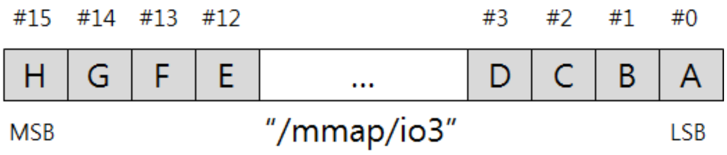
9.2.4 UDP

Product	Path
PBH-101, PBH-104, PBH-204	/mmap/udp0
	/mmap/udp1
	/mmap/udp2
	/mmap/udp3
	/mmap/udp4

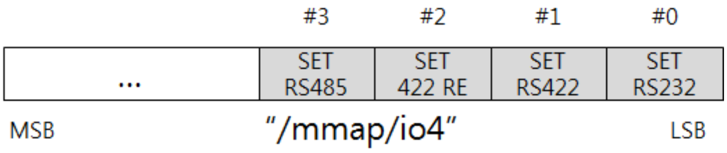
Table 9-5 UDP

9.2.5 I/O

- PBH-101

<table>
<tr><th colspan="2">Division</th><th>Path and Mapping Information</th></tr>
<tr><td rowspan="2">PBH-101</td><td>LED</td><td>/mmap/io3


#15
#14
#13
#12
#3
#2
#1
#0
H
G
F
E
…
D
C
B
A
MSB
"/mmap/io3"
LSB

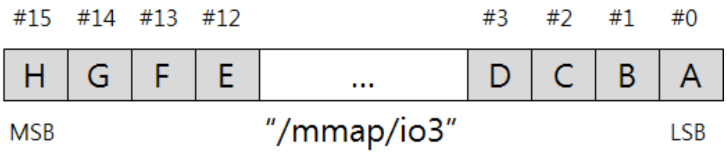
</td></tr>
<tr><td>UART Mode
(Serial Type)</td><td>/mmap/io4


#3
#2
#1
#0
…
SET RS485
SET 422 RE
SET RS422
SET RS232
MSB
"/mmap/io4"
LSB

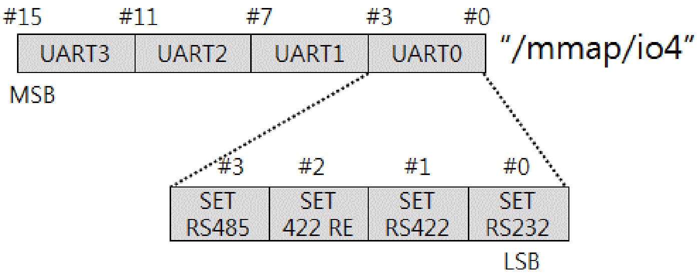
● example of setting UART mode
<table><tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr><tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table></td></tr>
</table>

Table 9-6 digital I/O of PBH-101

- PBH-104

<table>
<tr><th colspan="2">Division</th><th>Path and Mapping Information</th></tr>
<tr><td rowspan="2">PBH-104</td><td>LED</td><td>/mmap/io3


#15
#14
#13
#12
#3
#2
#1
#0
H
G
F
E
...
D
C
B
A
MSB
"/mmap/io3"
LSB

</td></tr>
<tr><td>UART Mode
(Serial Type)</td><td>/mmap/io4


#15
#11
#7
#3
#0
UART3
UART2
UART1
UART0
"/mmap/io4"
MSB
#3
#2
#1
#0
SET RS485
SET 422 RE
SET RS422
SET RS232
LSB

● example of setting UART mode
<table>
<tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr>
<tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table>
● example of setting multiple UART
<table>
<tr><th>Mode</th><th>Value</th><th>UART0</th><th>UART1</th><th>UART2</th><th>UART3</th></tr>
<tr><td>RS232</td><td>0x5555</td><td>0x0005</td><td>0x0050</td><td>0x0500</td><td>0x5000</td></tr>
<tr><td>RS422</td><td>0x2222</td><td>0x0002</td><td>0x0020</td><td>0x0200</td><td>0x2000</td></tr>
<tr><td>RS485</td><td>0xCCCC</td><td>0x000C</td><td>0x00C0</td><td>0x0C00</td><td>0xC000</td></tr>
</table>
</td></tr>
</table>

Table 9-7 digital I/O of PBH-104

- PBH-204

<table>
<tr><th colspan="2">Division</th><th>Path and Mapping Information</th></tr>
<tr><td rowspan="4">PBH-204</td><td>LED</td><td>/mmap/io3
<table><tr><td>#15</td><td>#14</td><td>#13</td><td>#12</td><td></td><td>#3</td><td>#2</td><td>#1</td><td>#0</td></tr><tr><td>H</td><td>G</td><td>F</td><td>E</td><td>...</td><td>D</td><td>C</td><td>B</td><td>A</td></tr></table>MSB "/mmap/io3" LSB</td></tr>
<tr><td>Digital Input (photo-coupler)</td><td>/mmap/io4
<table><tr><td>#15</td><td>#14</td><td>#13</td><td>#12</td><td>#11 ... #0</td></tr><tr><td>Di3</td><td>Di2</td><td>Di1</td><td>Di0</td><td>...</td></tr></table>MSB "/mmap/io4" LSB</td></tr>
<tr><td>Digital Output (relay)</td><td>/mmap/io4
<table><tr><td>#15 ... #12</td><td>#11</td><td>#10</td><td>#9</td><td>#8</td><td>#7</td><td>#6 ... #0</td></tr><tr><td>...</td><td>Do3</td><td>Do2</td><td>Do1</td><td>Do0</td><td>OE</td><td>...</td></tr></table>MSB "/mmap/io4" LSB
※ OE: bit for enabling or disabling output relay
- Enable: LOW(0), Disable: HIGH(1)</td></tr>
<tr><td>UART Mode</td><td>/mmap/io4
<table><tr><td></td><td>#3</td><td>#2</td><td>#1</td><td>#0</td></tr><tr><td>...</td><td>SET RS485</td><td>SET 422 RE</td><td>SET RS422</td><td>SET RS232</td></tr></table>MSB "/mmap/io4" LSB
● example of setting UART mode
<table><tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr><tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table></td></tr>
</table>

Table 9-8 digital I/O of PBH-204

9.2.6 ST

Product	Path
PBH-101, PBH-104, PBH-204	/mmap/st0
	/mmap/st1
	/mmap/st2
	/mmap/st3
	/mmap/st4
	/mmap/st5
	/mmap/st6
	/mmap/st7

Table 9-9 ST

9.2.7 ENV and User Memory

Division		Path	Size (Byte)
PBH-101, PBH-104, PBH-204	System ENV	/mmap/envs	1536
	User ENV	/mmap/envu	1536
	User Memory	/mmap/um0	64
		/mmap/um1	64
		/mmap/um2	64
		/mmap/um3	64

Table 9-10 ENV and user memory

10 Appendix: F/W Specification and Restriction

10.1 Firmware

Firmware	Product
P20	PBH-101, PBH-104, PBH-204

Table 10-1 firmware

10.2 Specification

Item	p20	Description
ENVS	1,536	Size of System ENV, byte
ENVU	1,536	Size of User ENV, byte
WLAN	1	Wireless LAN
EMAC	1	Ethernet
UART	4	The number of UART
FLOAT	Support	Floating Point Numbers
SSL	Support	SSL communication
PHP_MAX_NAME_SPACE	16	The number of Namespace
PHP_NAME_LEN	32	Size of User Identifier
PHP_MAX_USER_DEF_NAME	480	The number of User Identifier
PHP_LLSTR_BLK_SIZE	64	Size of String Block, byte
PHP_MAX_LLSTR_BLK	192	The number of String Blocks
string buffer size	12K	Size of string buffer, byte
PHP_MAX_STRING_LEN	1,536	Size of string variable, byte
PHP_INT_MAX	$\fallingdotseq 9.2*10^{18}$	Max value of integer type
EZFS_MAX_NAME_LEN	64	Size of EZFS filename, byte
TASK	2	The number of Task
TCP	5	The number of TCP
UDP	5	The number of UDP
TCP_RXBUF_SIZE	1,068	TCP receive buffer size
TCP_TXBUF_SIZE	1,152	TCP send buffer size
PDB_TXBUF_SIZE	2,048	PHPoCD send buffer size
HTTP_TXBUF_SIZE	1,536	HTTP send buffer size
UART_RXBUF_SIZE	1,024	UART send/receive buffer size
UDP_RXBUF_SIZE	512	UDP receive buffer size
ST	8	Software Timer

Table 10-2 firmware specification

10.3 Limitations

Item	limitation
Level of Namespace	PHP_MAX_NAME_SPACE - 1
Level of Function Call	PHP_MAX_NAME_SPACE - 2
Size of User Identifier	PHP_NAME_LEN - 1
Size of String Variable	PHP_MAX_STRING_LEN - 2
Size of Array Offset	string length - 2
Size of Filename	EZFS_MAX_NAME_LEN - 1
Size of arguments for system function	PHP_LLSTR_BLK_SIZE - 1
Size of arguments for pid_ioctl function	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function sendto	PHP_LLSTR_BLK_SIZE - 1
Size of $needle & $replace of function str_replace	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function inet_pton	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function inet_ntop	PHP_LLSTR_BLK_SIZE - 1
Size of $delimiter of function explode	PHP_LLSTR_BLK_SIZE - 1
Maximum size of UDP data for receiving	UDP receive buffer size - 2

Table 10-3 limitations

11 pid_ioctl Command Index

Device	Cmd.	Argument / Value	Page
NET	get	mode	- 20 -
	get	speed	- 20 -
	get	hwaddr	- 20 -
	get	ipaddr	- 20 -
	get	netmask	- 20 -
	get	gwaddr	- 20 -
	get	nsaddr	- 20 -
ST	get	count	- 41 -
	get	repc	- 44 -
	get	state	- 39 -
	set	div us/ms/sec	- 39 -
	set	mode free	- 39 -
	set	mode output toggle	- 39 -
	set	mode output pulse	- 39 -
	set	mode output pwm	- 39 -
	set	count (int)	- 41 -
	set	count (int1) [(int2) … (int8)]	- 44 -
	set	count (int1) (int2)	- 50 -
	set	dir up/down	- 41 -
	set	repc (int)	- 44 -
	set	delay (int)	- 44 -
	set	output low/high	- 44 -
	set	output invert [0/1]	- 44 -
	set	output dev io3/4 (int)	- 44 -
	set	trigger from php/st0/st1…/st7	- 44 -
	reset		- 39 -
	start		- 39 -
	stop		- 39 -
TCP	get	state	- 28 -
	get	rxlen	- 28 -
	get	rxbuf	- 28 -
	get	txfree	- 28 -
	get	txbuf	- 28 -
	get	dstport	- 28 -
	get	srcport	- 28 -

	get	dstaddr	- 28 -
	get	srcaddr	- 28 -
	get	ssh username	- 28 -
	get	ssh password	- 28 -
	set	nodelay 0/1	- 22 -
	set	api telnet/ssl/ssh/ws	- 22 -
	set	ssl method ssl3_client/ssl3_server/tls1_client/tls1/server	- 22 -
	set	ssh auth accept/reject	- 22 -
	set	ws path/mode/proto/origin	- 22 -
UART	get	rxlen	- 14 -
	get	txfree	- 14 -
	get	rxbuf	- 14 -
	get	txbuf	- 14 -
	get	flowctrl	- 14 -
	get	baud	- 14 -
	get	parity	- 14 -
	get	data	- 14 -
	get	stop	- 14 -
	set	baud (int)	- 12 -
	set	parity 0/1/2/3/4	- 12 -
	set	data 7/8	- 12 -
	set	stop 1/2	- 12 -
	set	flowctrl 0/1/2/3	- 12 -
UDP	get	rxlen	- 35 -
	get	dstport	- 35 -
	get	srcport	- 35 -
	get	dstaddr	- 35 -
	get	srcaddr	- 35 -
	set	dstaddr (string)	- 34 -
	set	dstport (int)	- 34 -
IO3/4	get	n mode	- 8 -
	get	n input/output	- 8 -
	set	n mode in/out/led_xx [low/high]	- 7 -
	set	n output low/high/toggle	- 7 -
	set	n lock/unlock	- 7 -

Table 11-1 pid_ioctl command index

12 Revision History

Date	Version	Note	Author
2014.09.23	1.0	○ Initial release	Roy LEE
2015.08.07	1.1	○ Change Document Name: add "for P20" ○ Add TCP APIs: TELNET, SSH, Websocket ○ Add ST output mode ○ Improve Appendix ○ Correct some errors and expressions	Roy LEE
2015.11.03	1.2	○ Correct some errors and expressions	Roy LEE