PHPoC

Device Programming Guide for P20

Version 1.2

솔내시스템㈜

PHPoC 포럼: http://www.phpoc.com/kr

홈페이지: http://www.sollae.co.kr

목차

1 개요

1.1 디바이스

PHPoC가 제공하는 하드웨어 장치 또는 소프트웨어 기능을 디바이스라고 합니다. 모든 디바이스들은 특수한 파일 형태로 제공되며 일반적인 파일 입/출력과 같은 방식으로 사용할 수 있습니다.

PHPoC의 파일 시스템 구조는 아래 그림과 같습니다.

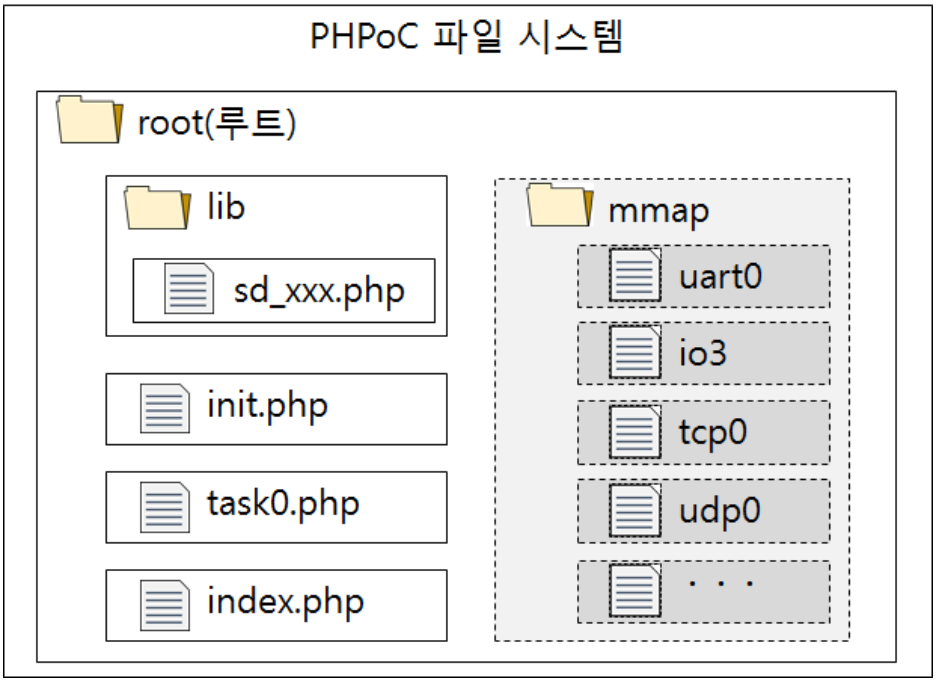


그림 1-1 PHPoC의 파일 시스템 구조

1.2 디바이스 파일의 위치

PHPoC에서 제공하는 모든 디바이스 파일들은 root 디렉터리 안에 있는 mmap(memory map)이라는 디렉터리에 위치합니다. 따라서 특정 디바이스는 다음과 같은 경로를 사용하여 접근해야 합니다.

```
/mmap/디바이스이름
```

☞ ***사용자가 PHPoC로 업로드 하는 모든 파일들은 파일 시스템의 최 상위 디렉터리인 root 디렉터리에 위치하게 됩니다. PHPoC에서 사용자가 접근할 수 있는 디렉터리로는 root와 /mmap이 있습니다. 사용자는 파일시스템에 임의로 디렉터리를 만들거나 삭제할 수 없습니다.***

1.3 디바이스 종류

PHPoC는 다음과 같은 디바이스를 제공합니다.

구분	디바이스 이름
하드웨어	디지털I/O(입/출력), UART(시리얼), NET(네트워크)
소프트웨어	TCP, UDP, ST(타이머)

표 1-1 디바이스 종류

제공되는 디바이스들의 개수는 제품 또는 펌웨어 버전에 따라서 달라질 수 있습니다.

☞ ***제품 별 디바이스에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

1.4 디바이스 사용 절차

일반적인 디바이스 사용법은 다음과 같습니다.

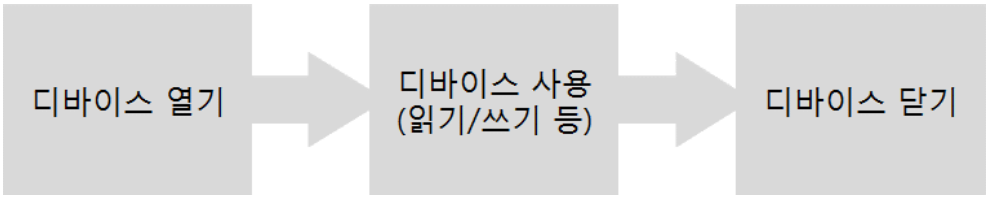


그림 1-2 디바이스 사용 절차

1.4.1 디바이스 열기

pid_open함수를 이용하여 각 디바이스를 열 수 있습니다. 이 함수는 pid(Peripheral ID)라는 정수 값을 반환하는데 이 값은 해당 디바이스에 접근할 수 있는 고유번호로 사용됩니다.

1.4.2 디바이스 사용

디바이스를 성공적으로 연 다음에는 반환 된 pid가 가리키는 디바이스를 사용할 수 있습니다.

1.4.3 디바이스 닫기

디바이스의 사용이 끝나서 더 이상 필요하지 않으면 해당 pid가 가리키는 디바이스를 pid_close함수를 이용해 닫습니다.

☞ ***디바이스에 따라서 pid_close함수가 필요하지 않을 수 있습니다.***

2 디지털 I/O

2.1 개요

디지털 I/O는 디지털 입력을 감시하거나 출력을 제어하는 용도로 사용할 수 있습니다. 또한 제품의 각종 상태를 나타내기 위한 LED로 연결하거나 시리얼통신 종류를 설정 할 때에도 사용됩니다.

- 디지털 I/O의 구조

모든 디지털 I/O의 상태는 HIGH(또는 1), LOW(또는 0)의 두 가지 상태 값을 가집니다. 따라서 각각의 포트는 다음과 같이 하나의 비트로 맵핑되어 표현됩니다.

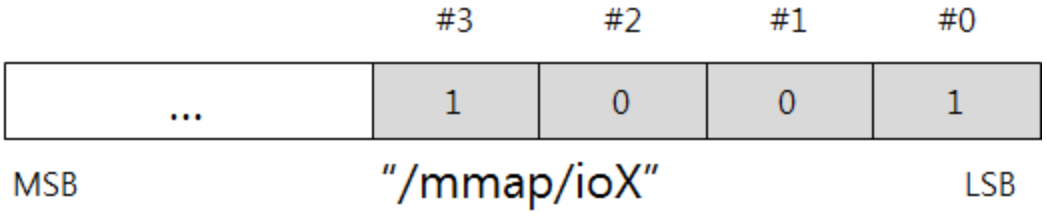


그림 2-1 디지털 I/O 맵핑 예

2.2 사용 절차

일반적인 디지털 I/O 사용 절차는 다음과 같습니다.

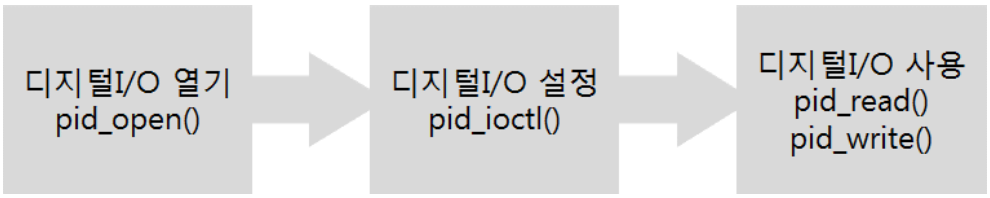


그림 2-2 디지털 I/O 사용 절차

2.3 디지털 I/O 열기

디지털 I/O를 열기 위해서는 pid_open함수를 사용합니다.

```
$pid = pid_open("/mmap/io3");          // 3번 디지털 I/O 열기
```

☞ ***LED를 포함한 제품 별 디지털 I/O에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

2.4 디지털 I/O 설정

디지털 I/O를 사용하기 전에 반드시 어떤 용도로 사용할 것인지를 설정 해야 합니다. 설정을 위해서는 pid_ioctl함수의 set명령을 사용합니다.

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

N1과 N2는 설정할 디지털 I/O의 포트번호의 범위를 나타냅니다. 단일 포트를 설정하는 경우에는 N2를 생략할 수 있습니다. 설정 가능한 TYPE은 다음과 같습니다.

2.4.1 설정 가능한 디지털 I/O 종류

TYPE		설명
in		디지털 입력
out	-	디지털 출력
	low	디지털 출력: 초기 값 LOW
	high	디지털 출력: 초기 값 HIGH
led_sys		시스템 상태 LED
led_net0_act / led_net1_act		NET(net0 - 유선, net1 - 무선)링크 활성화 LED: - NET이 네트워크에 정상적으로 연결된 경우 LOW - 네트워크 데이터를 송신 또는 수신하는 순간 HIGH
led_net0_link / led_net1_link		NET 링크 LED: 네트워크 연결 시 LOW
led_net0_rx / led_net1_rx		NET 수신 LED: 네트워크로부터 데이터를 수신하는 순간 LOW
led_net0_tx / led_net1_tx		NET 송신 LED: 네트워크로 데이터를 송신하는 순간 LOW

표 2-1 설정 가능한 디지털 I/O 종류

☞ ***LED가 인터페이스 되어있는 제품들은 위 포트들을 물리적인 LED로 맵핑할 수 있습니다. 물리적인 LED는 포트 상태가 LOW이면 켜지고(ON) HIGH이면 꺼집니다(OFF). 자세한 내용은 해당 제품의 회로도를 참조하시기 바랍니다.***

- 디지털 I/O종류 설정 예

```
$pid = pid_open("/mmap/io3");                  // 3번 디지털 I/O 열기
pid_ioctl($pid, "set 0 mode out");             // 0번 출력(out)포트로 설정
pid_ioctl($pid, "set 1-2 mode out high");      // 1~2번 출력(in)포트로 설정: HIGH
pid_ioctl($pid, "set 3 mode led_net0_link");   // 3번 NET 링크LED로 설정
pid_ioctl($pid, "set 12 mode led_net0_rx");    // 12번 NET 수신 LED로 설정
pid_ioctl($pid, "set 13 mode led_net0_tx");    // 13번 NET 송신 LED로 설정
```

2.5 디지털 I/O 사용

2.5.1 디지털 I/O 상태 읽기

디지털 I/O 상태를 읽을 때에는 pid_read함수를 사용하여 모든 포트의 상태를 한 번에 읽거나, pid_ioctl함수의 get명령을 사용해 단일 포트 정보를 읽습니다. 단일 포트 정보의 경우에는 입/출력 상태뿐만 아니라 설정 유형까지 읽을 수 있습니다.

```
pid_read($pid, VALUE);                  // 모든 포트 상태 읽기(16 bits 단위)
pid_ioctl($pid, "get N ITEM);           // 단일 포트 정보 읽기(1 bit 단위)
```

단일 포트 정보 읽기에서 사용 가능한 ITEM은 다음과 같습니다.

ITEM	설명	
mode	해당 포트의 설정 유형을 문자열 형태로 반환	입/출력 핀: "in", "out", "led_xxx" 등
		ST 출력으로 사용중인 핀: "st_out"
input	해당 입력포트의 상태를 정수 형태로 반환 (0: LOW, 1: HIGH)	
output	해당 출력포트의 상태를 정수 형태로 반환 (0: LOW, 1: HIGH)	

표 2-2 단일 포트 정보 읽기에서 사용 가능한 ITEM

● 모든 포트 상태 읽기 예

아래 예제는 3번 IO의 포트들을 입력포트로 설정한 후 그 상태 값을 읽어서 출력합니다.

```
$value = 0;
$pid = pid_open("/mmap/io3");          // 3번 디지털 I/O 열기
pid_ioctl($pid, "set 0-3 mode in");    // 0~3번 입력 설정
pid_read($pid, $value);                // 디지털 I/O 상태 읽기(16bits 단위)
printf("0x%x\r\n", $value);            // 출력 결과 예: 0xf00f
```

● 단일 포트 정보 읽기 예

아래 예제는 3번 IO의 0번 포트를 기본 상태가 HIGH인 출력포트로 설정하고 설정 유형 및 출력 상태를 읽어서 출력합니다.

```
$pid = pid_open("/mmap/io3");              // 디지털 I/O 열기
pid_ioctl($pid, "set 0 mode out high");    // 0번 포트 출력 설정(HIGH)
$mode = pid_ioctl($pid, "get 0 mode");     // 0번 포트 설정 유형 확인
$output = pid_ioctl($pid, "get 0 output"); // 0번 포트 출력상태 확인
printf("%s, %d\r\n", $mode, $output);      // 출력 결과: out, 1
```

☞ ***pid_ioctl의 get명령으로 단일포트의 상태를 읽을 때 해당 포트의 유형이 입력포트인 경우에는 "get N input"을, 출력포트인 경우에는 "get N output"을 사용해야 합니다.***

2.5.2 디지털 I/O 에 값 쓰기

디지털 I/O에 값을 출력하기 위해서는 pid_write함수를 사용하여 모든 포트에 한번에 출력하거나, pid_ioctl함수의 set명령을 사용해 단일 포트로 출력합니다.

```
pid_write($pid, VALUE);                  // 모든 포트 출력(16 bits 단위)
pid_ioctl($pid, "set N output TYPE");    // 단일 포트 출력(1 bit 단위)
```

● 모든 포트 출력 예

아래 예제는 3번 IO의 모든 포트를 출력포트로 설정한 후 임의의 값을 쓰고, 다시 디지털 I/O의 상태를 읽어 출력하는 예제 입니다.

```
$value = 0;
$pid = pid_open("/mmap/io3");                  // 3번 디지털 I/O 열기
pid_ioctl($pid, "set 0-7 mode out");           // 디지털 I/O 0~7번 출력 설정
pid_read($pid, $value);                        // 디지털 I/O 상태 읽기
pid_write($pid, ($value & 0xff00) | 0x0055); // 0x0055 출력
pid_read($pid, $value);                        // 디지털 I/O 상태 읽기
printf("0x%0x\r\n", $value);                   // 출력 결과 예: 0x0055
```

● 단일 포트 출력 예

아래 예제는 3번 IO의 0번포트를 기본 상태가 LOW인 출력포트로 설정하고 HIGH를 출력한 후 입/출력 상태를 읽어서 출력합니다.

```
$pid = pid_open("/mmap/io3");                  // 3번 디지털 I/O 열기
pid_ioctl($pid, "set 0 mode out low");         // 0번 포트 출력 설정(LOW)
pid_ioctl($pid, "set 0 output high");          // 0번 포트에 HIGH 출력
$output = pid_ioctl($pid, "get 0 output");     // 0번 포트 출력상태 확인
printf("%d\r\n", $output);                     // 출력 결과: 1
```

● 출력 제한 설정 예

아래 예제는 0번 포트에 출력 제한을 설정한 경우와 그렇지 않은 경우에 HIGH 출력의 적용 여부를 비교하는 예제 입니다.

```
$pid = pid_open("/mmap/io3");                  // 디지털 I/O 열기
pid_ioctl($pid, "set 0 mode out low");         // 0번 포트 출력 설정(LOW)
pid_ioctl($pid, "set 0 lock");                 // 0번 포트 출력 제한 설정
pid_ioctl($pid, "set 0 output high");          // 0번 포트에 HIGH 출력
$output1 = pid_ioctl($pid, "get 0 output"); // 0번 포트 출력상태 확인
pid_ioctl($pid, "set 0 unlock");               // 0번 포트 출력 제한 해제
pid_ioctl($pid, "set 0 output high");          // 0번 포트에 HIGH 출력
$output2 = pid_ioctl($pid, "get 0 output");    // 0번 포트 출력상태 확인
printf("%d, %d\r\n", $output1, $output2);      // 출력 결과: 0, 1
```

2.5.3 릴레이 제어하기

디지털 출력포트가 릴레이로 연결 된 외장형 제품들이 있습니다.

● 릴레이 제어 예

아래 예제는 릴레이 포트를 매 초 마다 0번부터 순서대로 ON한 다음 모든 포트를 ON/OFF하는 예제 입니다.

```
$pid = pid_open("/mmap/io4");          // 4번 디지털 I/O(릴레이) 열기
pid_ioctl($pid, "set 7 mode out");     // 7번(릴레이 활성)을 출력으로 설정
pid_ioctl($pid, "set 8-11 mode out");  // 8 ~ 11번 설정(릴레이 포트 0 ~ 3번)
pid_write($pid, 0x0100);               // 릴레이 포트 0번 ON
sleep(1);
pid_write($pid, 0x0200);               // 릴레이 포트 1번 ON
sleep(1);
pid_write($pid, 0x0400);               // 릴레이 포트 2번 ON
sleep(1);
pid_write($pid, 0x0800);               // 릴레이 포트 3번 ON
sleep(1);
pid_write($pid, 0x0f00);               // 릴레이 포트 0 ~ 3번 모두 ON
sleep(1);
pid_write($pid, 0x0000);               // 릴레이 포트 0 ~ 3번 모두 OFF
```

☞ ***위 예제는 릴레이 포트가 없는 제품의 경우에는 사용할 수 없습니다.***

3 UART

UART(Universal Asynchronous Receiver and Transmitter)는 가장 널리 사용되고 있는 시리얼 통신 방식입니다.

3.1 사용 절차

일반적인 UART 사용 절차는 다음과 같습니다.

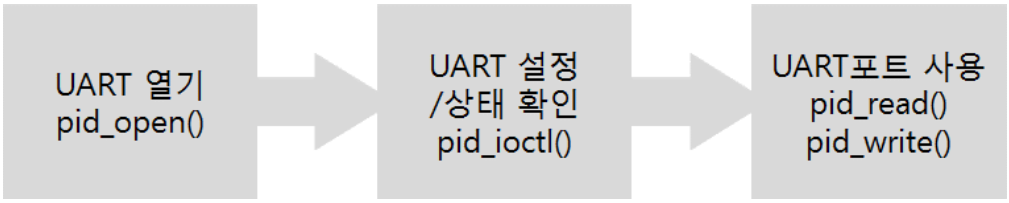


그림 3-1 UART 사용 절차

3.2 UART 열기

UART를 열기 위해서는 pid_open함수를 사용합니다.

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

☞ ***제품 별 UART에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

3.3 UART 설정

UART를 사용하기 전에 반드시 설정이 필요합니다. 통신속도(baud rate), 데이터비트(data bit), 정지 비트(stop bit), 패리티(parity)등의 설정 항목이 있으며, 설정을 위해서는 pid_ioctl함수의 set명령을 사용합니다.

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

ITEM은 설정 할 항목을, VALUE는 항목에 설정할 값을 의미합니다.

3.3.1 설정 가능한 UART 항목

ITEM	VALUE	설명
baud	예) 9600	통신속도[bps], 2400 ~ 230400
parity	0	패리티 사용 안 함
	1	EVEN(짝수 패리티)
	2	ODD(홀수 패리티)
	3	MARK(패리티 비트 항상 1)
	4	SPACE(패리티 비트 항상 0)
data	8	데이터 비트 8
	7	데이터 비트 7(이 때 반드시 패리티를 사용해야 함)
stop	1	정지 비트 1
	2	정지 비트 2
flowctrl	0	흐름제어 사용 안 함
	1	RTS/CTS 흐름제어 사용
	2	Xon/Xoff 흐름제어 사용
	3	TxDE 흐름제어 사용(RS485)

표 3-1 설정 가능한 UART 항목

- UART 설정 예

```
$pid = pid_open("/mmap/uart0");      // 0번 UART 열기
pid_ioctl($pid, "set baud 9600");    // 통신속도 9600 bps
pid_ioctl($pid, "set parity 0");     // 패리티 사용 안 함
pid_ioctl($pid, "set data 8");       // 데이터 비트 8
pid_ioctl($pid, "set stop 1");       // 정지 비트 1
pid_ioctl($pid, "set flowctrl 0");   // 흐름제어 사용 안 함
```

3.3.2 시리얼 통신 방식 설정

UART는 제품에 따라 RS232뿐만 아니라 RS422이나 RS485로도 확장이 가능합니다. 시리얼통신 방식을 설정하기 위해서는 TxDE 및 관련 입/출력 핀을 알맞게 설정해야 합니다.

다음 예제는 0번 UART의 시리얼 통신 방식을 변경하는 방법을 안내하는 예제 입니다.

```
$pid = pid_open("/mmap/uart0");              // 0번 UART 열기
pid_ioctl($pid, "set baud 9600");            // 통신속도 9600 bps
pid_ioctl($pid, "set parity 0");             // 패리티 사용 안 함
pid_ioctl($pid, "set data 8");               // 데이터 비트 8
pid_ioctl($pid, "set stop 1");               // 정지 비트 1
pid_ioctl($pid, "set flowctrl 3");           // TxDE 흐름제어 사용
$pid_mode = pid_open("/mmap/io4");           // UART모드 설정 포트 열기
pid_ioctl($pid_mode, "set 0-3 mode out");    // UART모드 설정 핀 초기화
pid_write($pid_mode, 0x05);                  // RS232 설정
//pid_write($pid_mode, 0x02);                // RS422 설정(주석 해제)
//pid_write($pid_mode, 0x0c);                // RS485 설정(주석 해제)
pid_close($pid_mode);
```

☞ ***위 예제는 "/mmap/io4"의 0-3번 비트가 시리얼 통신방식을 설정하는 용도로 맵핑된 경우를 가정하였습니다. 통신방식 설정을 위한 맵핑 경로 및 설정 값은 제품에 따라 다를 수 있으니 제품 별 디바이스 관련 정보를 확인하시기 바랍니다.***

☞ ***제품 별 디지털 I/O 관련 정보는 부록을 참조하시기 바랍니다.***

3.4 UART 상태정보 확인

pid_ioctl함수의 get명령어로 UART의 각종 상태를 확인 할 수 있습니다.

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

3.4.1 확인 가능한 UART 상태정보

ITEM	설명	반환 값	반환 형식
baud	통신속도[bps]	예) 9600	정수
parity	패리티	0 / 1 / 2 / 3 / 4	정수
data	데이터비트[bit]	8 / 7	정수
stop	정지비트[bit]	1 / 2	정수
flowctrl	흐름제어	0 / 1 / 2 / 3	정수
txbuf	송신버퍼 크기[Byte]	예) 1024	정수
txfree	송신버퍼 여유공간[Byte]	예) 1024	정수
rxbuf	수신버퍼 크기[Byte]	예) 1024	정수
rxlen	수신 데이터 크기[Byte]	예) 10	정수

표 3-2 확인 가능한 UART 상태정보

- UART 상태정보 확인 예

UART의 현재 설정 값은 다음과 같이 확인 할 수 있습니다.

```
$pid = pid_open("/mmap/uart0");                 // 0번 UART 열기
$baud = pid_ioctl($pid, "get baud");            // 통신속도
$parity = pid_ioctl($pid, "get parity");        // 패리티
$data = pid_ioctl($pid, "get data");            // 데이터비트
$stop = pid_ioctl($pid, "get stop");            // 정지비트
$flowctrl = pid_ioctl($pid, "get flowctrl");    // 흐름제어
echo "baud = $baud\r\n";                        // 출력 예: baud = 9600
echo "parity = $parity\r\n";                    // 출력 예: parity = 0
echo "data = $data\r\n";                        // 출력 예: data = 8
echo "stop = $stop\r\n";                        // 출력 예: stop = 1
echo "flowctrl = $flowctrl\r\n";                // 출력 예: flowctrl = 0
```

3.4.2 송신버퍼에 남아있는 데이터 크기

UART의 송신버퍼에 남아있는 데이터 크기는 다음과 같이 계산할 수 있습니다.

```
송신버퍼에 남아있는 데이터 크기 = 송신버퍼 크기 - 송신버퍼 여유공간
```

- 사용 예

이 예제는 0번 UART에 10바이트를 전송하고, 송신버퍼에 남아있는 데이터 크기를 계산하여 출력합니다.

```
$txlen = -1;
$data = "0123456789";
$pid = pid_open("/mmap/uart0");       // 0번 UART 열기
pid_ioctl($pid, "set baud 9600");     // 통신속도 9600 bps
pid_ioctl($pid, "set parity 0");      // 패리티 사용 안 함
pid_ioctl($pid, "set data 8");        // 데이터 비트 8
pid_ioctl($pid, "set stop 1");        // 정지 비트 1
pid_ioctl($pid, "set flowctrl 0");    // 흐름제어 사용 안 함
pid_write($pid, $data);               // UART에 데이터($data) 송신
while($txlen)
{
        $txbuf = pid_ioctl($pid, "get txbuf");   // 송신버퍼 크기 확인
        $txfree = pid_ioctl($pid, "get txfree"); // 송신버퍼 여유공간 확인
        $txlen = $txbuf - $txfree;       // 송신버퍼에 남아있는 데이터 크기 계산
        echo "tx len = $txlen\r\n";      // 송신버퍼에 남아있는 데이터 크기 출력
        usleep(1000);
}
pid_close($pid);
```

3.4.3 수신 데이터 크기

UART가 수신한 데이터 크기는 다음과 같이 확인 할 수 있습니다.

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

- 특정 문자(열)까지 수신한 데이터 크기 확인하기

rxlen명령어 뒤에 특정 문자열($string)을 입력하면 pid_ioctl함수는 해당 문자열이 들어오기 전까지는 0을 반환하다가 해당 문자열이 들어오면 그 문자열까지의 수신 데이터 크기를 반환합니다.

3.4.4 수신버퍼 여유공간

UART의 수신버퍼 여유공간은 다음과 같이 계산할 수 있습니다.

```
수신버퍼 여유공간 = 수신버퍼 크기 - 수신 데이터 크기
```

- 사용 예

이 예제는 0번 UART의 수신버퍼의 여유공간을 계산하여 출력합니다.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");      // 0번 시리얼포트 열기
pid_ioctl($pid, "set baud 9600");       // 통신속도 9600 bps
pid_ioctl($pid, "set parity 0");          // 패리티 사용 안 함
pid_ioctl($pid, "set data 8");             // 데이터 비트 8
pid_ioctl($pid, "set stop 1");             // 정지 비트 1
pid_ioctl($pid, "set flowctrl 0");         // 흐름제어 사용 안 함
$rxbuf = pid_ioctl($pid, "get rxbuf");  // 수신버퍼 크기 확인
$rxlen = pid_ioctl($pid, "get rxlen");  // 수신 데이터 크기 확인
$rxfree = $rxbuf - $rxlen;                // 수신버퍼 여유공간 계산
echo "rxfree = $rxfree\r\n";              // 수신버퍼 여유공간 출력
pid_close($pid);
```

3.5 UART 사용

3.5.1 데이터 수신

시리얼포트로부터 들어온 데이터는 수신버퍼에 저장됩니다. 이 수신버퍼에 저장 된 값을 pid_read함수로 읽어서 데이터를 수신합니다.

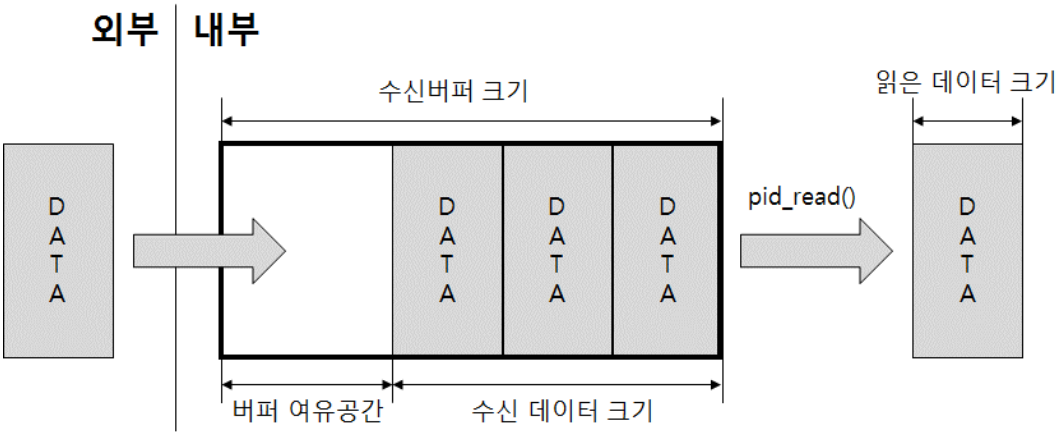


그림 3-2 UART 데이터 수신

pid_read함수는 다음과 같이 사용합니다.

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

$var는 읽은 값을 저장 할 변수이고, $len은 읽을 바이트 수를 의미합니다.

● 사용 예

이 예제는 약 1초마다 UART로 수신되는 데이터를 확인하여 출력합니다.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");                 // 0번 UART 열기
pid_ioctl($pid, "set baud 9600");               // 통신속도 9600bps
pid_ioctl($pid, "set parity 0");                // 패리티 사용 안 함
pid_ioctl($pid, "set data 8");                  // 데이터 비트 8
pid_ioctl($pid, "set stop 1");                  // 정지 비트 1
$rxbuf = pid_ioctl($pid, "get rxbuf");          // 수신버퍼 크기 확인
while(1)
{
        $rxlen = pid_ioctl($pid, "get rxlen");    // 수신 데이터 크기 확인
        $rx_free = $rxbuf - $rxlen;               // 버퍼 여유공간 확인
        echo "$rx_free / $rxbuf\r\n";             // 수신버퍼 여유공간/크기 출력
        $len = pid_read($pid, $rdata, $rxlen); // 데이터 읽기
        echo "len = $len / ";                     // 읽은 데이터 크기 출력
        echo "rdata = $rdata\r\n";                // 읽은 데이터 출력
        sleep(1);
}
pid_close($pid);
```

3.5.2 데이터 송신

pid_write함수를 이용해 쓰기 한 데이터는 송신버퍼에 저장되었다가 UART를 통해 외부로 송신됩니다.

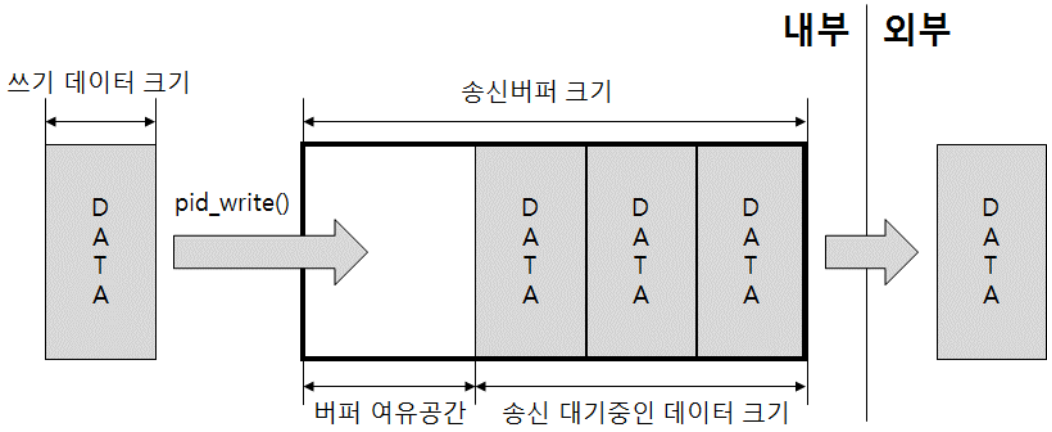


그림 3-3 UART 데이터 송신

pid_write함수는 다음과 같이 사용합니다.

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

$var는 쓸 데이터가 저장 된 변수이고, $wlen은 쓸 바이트 수 입니다.

● 사용 예

이 예제는 약 1초마다 송신버퍼의 여유공간을 확인하여 UART로 데이터를 출력합니다.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/uart0");                  // 0번 시리얼포트 열기
pid_ioctl($pid, "set baud 9600");                // 통신속도 9600bps
$txbuf = pid_ioctl($pid, "get txbuf");           // 송신버퍼 크기 확인
while(1)
{
        $txfree = pid_ioctl($pid, "get txfree");  // 송신버퍼 여유공간 확인
        echo "txfree = $txfree\r\n";              // 송신버퍼 여유공간 출력
        $len = pid_write($pid, $sdata, $txfree); // 데이터 쓰기
        echo "len = $len\r\n";                    // 쓰기 한 데이터 크기 출력
        sleep(1);
}
pid_close($pid);
```

위 코드에서 pid_write함수의 세 번째 인수는 쓰기 할 데이터의 크기를 의미합니다. 쓰기 데이터크기가 송신버퍼의 여유공간보다 크면 데이터가 유실될 수 있습니다. 따라서 항상 버퍼 여유공간을 확인한 후 그 값 또는 그 이하의 값으로 쓰기 데이터 크기를 설정하시기 바랍니다.

4 NET

NET는 물리적인 유/무선 네트워크 인터페이스입니다.

4.1 사용 절차

일반적인 NET 사용 절차는 다음과 같습니다.

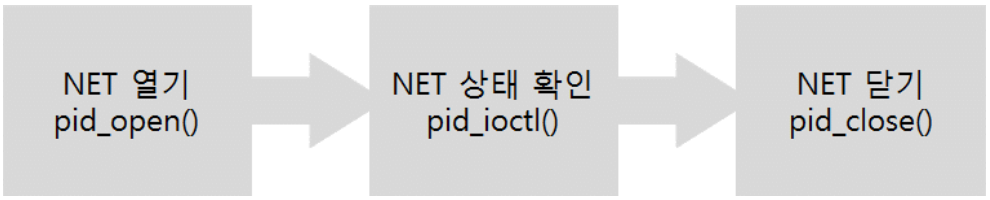


그림 4-1 NET 사용 절차

4.2 NET 열기

NET를 열기 위해서는 pid_open함수를 사용합니다.

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

☞ ***제품 별 NET에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

4.3 NET 상태정보 확인

pid_ioctl함수의 get명령어를 이용하여 NET의 상태정보를 확인할 수 있습니다.

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

ITEM은 확인 가능한 상태정보의 이름입니다.

4.3.1 확인 가능한 NET 상태정보

ITEM	설명	반환 값	반환 형식
hwaddr	MAC주소	예) 00:30:f9:00:00:01	문자열
ipaddr	설정 된 IP주소	예) 10.1.0.1	문자열
netmask	서브넷마스크	예) 255.0.0.0	문자열
gwaddr	게이트웨이 주소	예) 10.1.0.254	문자열
nsaddr	네임서버 주소	예) 10.1.0.254	문자열
mode	10M이더넷	10BASET	문자열
	100M이더넷	100BASET	문자열
	무선랜 사용 불가	""(빈 문자열)	문자열
	무선랜 인프라스트럭처	INFRA	문자열
	무선랜 애드혹	IBSS	문자열
	무선랜 Soft AP	AP	문자열
speed	유선랜 속도[Mbps]	0 / 10 / 100	정수
	무선랜 속도[100Kbps]	0 / 10 / 20 / 55 / 110 / 60 / 90 / 120 / 180 / 240 / 360 / 480 / 540	정수

표 4-1 확인 가능한 NET 상태정보

- NET 상태정보 확인 예

이 예제는 NET의 각 상태정보를 확인하여 출력합니다.

```
$pid = pid_open("/mmap/net0");                  // 0번 NET 열기
echo pid_ioctl($pid, "get hwaddr"), "\r\n";     // MAC주소 확인
echo pid_ioctl($pid, "get ipaddr"), "\r\n";     // IP주소 확인
echo pid_ioctl($pid, "get netmask"), "\r\n";    // 서브넷마크스 확인
echo pid_ioctl($pid, "get gwaddr"), "\r\n";     // 게이트웨이 주소 확인
echo pid_ioctl($pid, "get nsaddr"), "\r\n";     // 네임서버 주소 확인
echo pid_ioctl($pid, "get mode"), "\r\n";       // 이더넷 방식 확인
echo pid_ioctl($pid, "get speed"), "\r\n";      // 이더넷 속도 확인
pid_close($pid);                                // 네트워크포트 닫기
```

5 TCP

TCP는 네트워크상에서 데이터를 전송할 때 사용하는 프로토콜로 UDP와 더불어 오늘날의 인터넷에서 사용되는 TCP/IP프로토콜 중 전송을 담당하는 핵심적인 프로토콜 입니다. TCP는 접속과정이 있으며 데이터의 신뢰성을 보장하는 특징이 있습니다.

5.1 사용 절차

일반적인 TCP 사용 절차는 다음과 같습니다.

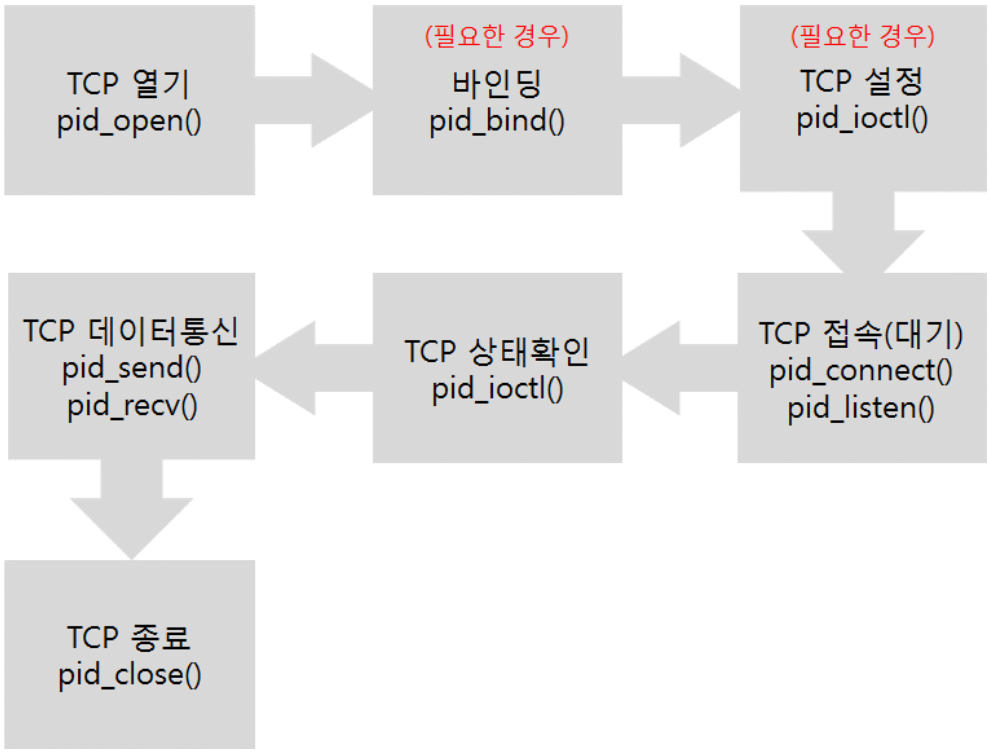


그림 5-1 TCP 사용 절차

☞ ***디바이스를 TCP서버로 동작 시키고자 하는 경우에는 바인딩 과정을 생략할 수 없습니다.***

5.2 TCP 열기

TCP를 열기 위해서는 pid_open함수를 사용합니다.

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

☞ ***제품 별 TCP에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

5.3 TCP 설정

TCP를 사용하기 전에 설정이 필요한 경우가 있습니다. 특히 SSL, SSH, TELNET 또는 웹 소켓을 사용하기 위해서는 pid_ioctl함수의 set명령을 이용하여 반드시 접속 전에 설정을 먼저 해 주어야 합니다.

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

ITEM은 설정 항목을, VALUE는 항목에 대한 값을 나타냅니다.

5.3.1 설정 가능한 TCP 항목

ITEM	VALUE		설명
nodelay	0		TCP Nagle 알고리즘 사용
	1		TCP Nagle 알고리즘 사용 안 함
api	ssl		SSL 사용
	ssh		SSH 서버 사용
	telnet		TELNET 서버 사용
	ws		웹 소켓(Web Socket) 서버 사용
ssl method	ssl3_client		SSL클라이언트 (SSL 3.0)
	tls1_client		SSL클라이언트 (TLS 1.0)
	ssl3_server		SSL서버 (SSL 3.0)
	tls1_server		SSL서버 (TLS 1.0)
ssh auth	accept		SSH 인증 허용
	reject		SSH 인증 거부
ws	path	PATH	웹 소켓 URI의 경로 설정
	mode	0	웹 소켓 데이터 형식: 텍스트(text)
		1	웹 소켓 데이터 형식: 바이너리(binary)
	proto	PROTOCOL	웹 소켓 통신에서 사용할 프로토콜
	origin	ADDR	접근을 허용 할 호스트의 주소 설정

표 5-1 설정 가능한 TCP 항목

TCP Nagle 알고리즘은 데이터를 전송할 때 세그먼트의 수를 최소한으로 줄여 전송 효율 높이기 위한 기능으로 약간의 지연시간이 동반됩니다.

☞ ***주의: "set api"명령으로 설정하는 항목들은 TCP0 ~ 3에서만 사용이 가능합니다. 또한 특정 TCP의 api모드가 설정 되면 해당 TCP는 제품이 리부팅 되기 전까지 다른 api모드로 사용할 수 없습니다.***

5.3.2 SSL 사용

PHPoC는 "set api ssl"명령을 이용해 SSL 서버 또는 클라이언트로 동작시킬 수 있습니다. 다음은 SSL 서버 설정 예 입니다.

- SSL 서버 설정 예

```
$port = 1470;                                       // 포트번호
$pid = pid_open("/mmap/tcp0");                      // 0번 TCP 열기
pid_ioctl($pid, "set api ssl");                     // SSL 사용
pid_ioctl($pid, "set ssl method tls1_server");      // SSL 서버(TLS 1.0) 설정
pid_bind($pid, "", $port);                          // 바인딩
pid_listen($pid);                                   // TCP 수동접속 대기
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);                            // TCP 접속 종료
}
```

☞ ***PHPoC를 SSL 서버로 사용할 때 제품에 인증서가 저장되어 있어야 합니다. 인증서는 PHPoC Debugger를 통해 생성하거나 저장할 수 있습니다.***

다음 예제는 PHPoC를 SSL클라이언트로 설정하는 예 입니다.

- SSL 클라이언트 설정 예

```
$addr = "10.1.0.2";                                 // 서버 IP주소
$port = 1470;                                       // 포트번호
$pid = pid_open("/mmap/tcp0");                      // 0번 TCP 열기
pid_ioctl($pid, "set api ssl");                     // SSL 사용
pid_ioctl($pid, "set ssl method tls1_client");      // SSL 서버(TLS 1.0) 설정
pid_connect($pid, $addr, $port);                    // TCP 능동접속 시도
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);                            // TCP 접속 종료
}
```

☞ ***PHPoC의 메모리 사용량이 늘어나면 SSL 통신을 위한 메모리 공간이 확보되지 못해 SSL동작이 원활하지 않거나 불가능 할 수 있습니다.***

5.3.3 TELNET 서버 사용

PHPoC를 "set api telnet"명령을 이용해 TELNET 서버로 동작시킬 수 있습니다. 다음은 TELNET 서버 설정 예 입니다.

- TELNET 서버 설정 예

```
$port = 23;                                          // 포트번호
$pid = pid_open("/mmap/tcp0");                       // 0번 TCP 열기
pid_ioctl($pid, "set api telnet");                   // TELNET 사용
pid_bind($pid, "", $port);                           // 바인딩
pid_listen($pid);                                    // TCP 수동접속 대기
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);                             // TCP 접속 종료
}
```

위 예제에서 PHPoC는 23번 포트로 접속을 대기하다가 TELNET 클라이언트가 접속하면 환영 메시지를 송신하고 접속을 종료합니다.

5.3.4 SSH 서버 사용

PHPoC를 "set api ssh"명령을 이용해 SSH 서버로 동작시킬 수 있습니다. 다음은 SSH 서버 설정 예 입니다.

- SSH 서버 설정 예

```
$port = 22;                                     // 포트번호
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_ioctl($pid, "set api ssh");                 // SSH 사용
pid_bind($pid, "", $port);                      // 바인딩
pid_listen($pid);                               // TCP 수동접속 대기
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;
        }
}
```

위 예제에서 PHPoC는 22번 포트로 접속을 대기하다가 SSH 클라이언트가 접속하면 사용자이름과 암호를 출력하여 인증을 허용합니다. 그리고는 환영 메시지를 송신하고 접속을 종료합니다.

☞ ***SSH의 사용자 아이디 및 암호 확인 과정은 PHPoC 스크립트에서 처리해야 합니다.***

5.3.5 웹 소켓 서버 사용

PHPoC를 "set api ws"명령을 이용해 웹 소켓 서버로 동작시킬 수 있습니다. 다음은 웹 소켓 서버 설정 예 입니다.

- 웹 소켓 서버 설정 예

이 예제는 웹 소켓이 접속 되고 클라이언트로부터 데이터를 수신하면 수신한 데이터를 출력 한 뒤 응답 횟수를 송신합니다.

```
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_ioctl($pid, "set api ws");                  // 웹 소켓 설정
pid_ioctl($pid, "set ws path test");            // URI 경로: /test
pid_ioctl($pid, "set ws mode 0");               // 전송 모드: 텍스트
pid_ioctl($pid, "set ws origin 10.1.0.1");      // 접근 허용 호스트: 10.1.0.1
pid_ioctl($pid, "set ws proto myproto");        // 프로토콜: myproto

pid_bind($pid, "", 0);                          // 바인딩: 기본포트(80)

$rwbuf = "";
$count = 1;

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

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

☞ ***웹 소켓 서버와 제품 기본 웹 서버 기능(index.php)을 활용하면 보다 더 효율적인 웹 인터페이스를 구현할 수 있습니다.***

☞ ***웹 소켓을 사용하기 위해서는 반드시 웹 소켓을 지원하는 웹 브라우저를 사용하시기 바랍니다.***

5.4 TCP 접속

5.4.1 TCP 클라이언트(능동 접속)

능동 접속은 접속을 대기하고 있는 TCP서버로 접속을 시도하는 것을 의미하며 능동 접속 호스트를 TCP클라이언트라고 합니다. 능동 접속을 위해서는 pid_connect함수를 사용해야 합니다.

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

여기서 $addr은 접속할 TCP서버의 IP주소를, $port는 TCP 포트번호를 의미합니다.

- TCP클라이언트 사용 예

```
$pid = pid_open("/mmap/tcp0");      // 0번 TCP 열기
$addr = "10.1.0.2";                 // 서버 IP주소
$port = 1470;                       // TCP포트
pid_connect($pid, $addr, $port);    // TCP능동접속 시도
sleep(25);
pid_close($pid);
```

5.4.2 TCP 서버(수동 접속)

수동 접속은 TCP클라이언트의 접속을 대기하고 있는 것을 의미하며 수동 접속 호스트를 TCP서버라고 합니다. 수동 접속을 위해서는 pid_bind함수와 pid_listen함수를 사용해야 합니다.

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

여기서 $port는 접속을 대기할 TCP 포트번호를 의미합니다.

- TCP서버 사용 예

```
$pid = pid_open("/mmap/tcp0");      // 0번 TCP 열기
$port = 1470;                       // TCP포트
pid_bind($pid, "", $port);          // 바인딩
pid_listen($pid);                   // TCP수동접속 대기
sleep(25);
pid_close($pid);
```

5.5 TCP 상태정보 확인

pid_ioctl함수의 get명령어로 TCP의 각종 상태를 확인 할 수 있습니다.

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

5.5.1 확인 가능한 TCP 상태정보

ITEM	설명	반환 값	반환 형식
state	TCP 세션 상태 - 접속 끊김	TCP_CLOSED	정수
	TCP 세션 상태 - 접속 완료	TCP_CONNECTED	정수
	TCP 세션 상태 - 접속 대기	TCP_LISTEN	정수
	SSL 세션 상태 - 접속 끊김	SSL_CLOSED	정수
	SSL 세션 상태 - 접속 완료	SSL_CONNECTED	정수
	SSL 세션 상태 - 접속 대기	SSL_LISTEN	정수
	SSH 세션 상태 - 접속 끊김	SSH_CLOSED	정수
	SSH 세션 상태 - 접속 완료	SSH_CONNECTED	정수
	SSH 세션 상태 - 접속 대기	SSH_LISTEN	정수
	SSH 세션 상태 - 인증 완료	SSH_AUTH	정수
srcaddr	장치의 로컬 IP주소	예) 192.168.0.1	문자열
srcport	장치의 로컬 TCP 포트번호	예) 1470	정수
dstaddr	TCP통신 상대방의 IP주소	예) 192.168.0.2	문자열
dstport	TCP통신 상대방의 포트번호	예) 1470	정수
txbuf	송신버퍼 크기[Byte]	예) 1152	정수
txfree	송신버퍼 여유공간[Byte]	예) 1152	정수
rxbuf	수신버퍼 크기[Byte]	예) 1068	정수
rxlen	수신 데이터 량[Byte]	예) 200	정수
ssh username	SSH 아이디	예) user	문자열
ssh password	SSH 비밀번호	예) password	문자열

표 5-2 확인 가능한 TCP 상태 정보

5.5.2 TCP 세션 상태

TCP는 접속과정 이후에 데이터통신을 하므로 TCP세션의 상태를 확인하는 것은 매우 중요합니다. 상태 값은 접속이 되지 않았거나 이미 끊긴 상태를 나타내는 TCP_CLOSED, 접속이 완료 된 상태인 TCP_CONNECTED, TCP서버로서 접속을 대기하고 있는 상태인 TCP_LISTEN의 세 가지 입니다. SSL과 SSH도 마찬가지로 접속 안 됨, 접속 완료 및 접속 대기의 세 가지 상태가 있으며 SSH는 인증(SSH_AUTH) 상태가 하나 더 있습니다. 모든 세션의 상태는 다음과 같이 확인할 수 있습니다.

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

☞ ***TCP, SSL 또는 SSH접속을 시도하는 중이거나 종료하는 중일 때 세션의 상태정보를 확인하는 경우 위 표에 나온 값들 이외의 값들이 반환 될 수 있습니다. 이 값들은 펌웨어 내부적으로만 사용되는 상수이며 향후 변경의 소지가 있으므로 사용자는 프로그래밍에 사용하지 마십시오.***

5.5.3 송신버퍼에 남아있는 데이터 크기

TCP 송신버퍼에 남아있는 데이터 크기는 다음과 같이 계산할 수 있습니다.

```
송신버퍼에 남아있는 데이터 크기 = 송신버퍼 크기 - 송신버퍼 여유공간
```

● 사용 예

이 예제는 TCP접속 후 8바이트를 전송하고 TCP 송신버퍼에 남아있는 데이터 크기를 계산하여 출력합니다.

```
$tx_len = -1;
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
do
{
        pid_connect($pid, "10.1.0.2", 1470);  // TCP 능동접속
        usleep(500000);
}
while(pid_ioctl($pid, "get state") != TCP_CONNECTED);
pid_send($pid, "01234567");                     // 8바이트 전송
while($tx_len && (pid_ioctl($pid, "get state") == TCP_CONNECTED))
{
        $txbuf = pid_ioctl($pid, "get txbuf");  // 송신버퍼 크기 확인
        $txfree = pid_ioctl($pid, "get txfree"); // 송신버퍼 여유공간 확인
        $tx_len = $txbuf - $txfree;      // 송신버퍼에 남아있는 데이터 크기 확인
        echo "tx len = $tx_len\r\n";     // 송신버퍼에 남아있는 데이터 크기 출력
        usleep(10000);
}
pid_close($pid);                                // TCP 종료
```

5.5.4 수신 데이터 크기

TCP에서 수신한 데이터 크기는 다음과 같이 확인 할 수 있습니다.

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

- 특정 문자(열)까지 수신한 데이터 크기 확인하기

rxlen명령어 뒤에 특정 문자열($string)을 입력하면 pid_ioctl함수는 해당 문자열이 들어오기 전까지는 0을 반환하다가 해당 문자열이 들어오면 그 문자열까지의 수신 데이터 크기를 반환합니다.

5.5.5 수신버퍼 여유공간

TCP 수신버퍼 여유공간은 다음과 같이 계산할 수 있습니다.

```
수신버퍼 여유공간 = 수신버퍼 크기 - 수신 데이터 크기
```

- 사용 예

이 예제는 TCP접속 후 수신버퍼의 여유공간을 계산하여 출력합니다.

```
$rx_free = 1068;
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
do
{
        pid_connect($pid, "10.1.0.2", 1470);   // TCP 능동접속
        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");    // 수신버퍼 크기 확인
        $rxlen = pid_ioctl($pid, "get rxlen");    // 수신 데이터 크기 확인
        $rx_free = $rxbuf - $rxlen;               // 수신버퍼 여유공간 확인
        echo "rx free = $rx_free\r\n";            // 수신버퍼 여유공간 출력
        sleep(1);
}
pid_close($pid);                                  // TCP 종료
```

5.6 TCP 데이터 통신

5.6.1 TCP 데이터 수신

네트워크로부터 들어온 TCP데이터는 수신버퍼에 저장됩니다. 이 수신버퍼에 저장 된 값을 pid_recv함수로 읽습니다.

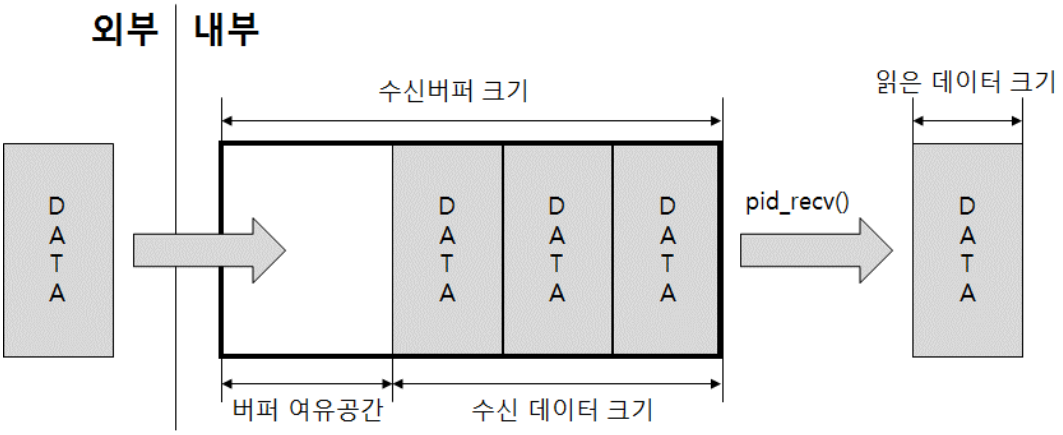


그림 5-2 TCP 데이터 수신

pid_recv함수는 다음과 같이 사용합니다.

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

● 사용 예

이 예제는 매 초마다 TCP로 수신되는 데이터를 확인하고 출력합니다.

```
$rdata = "";
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_connect($pid, "10.1.0.2", 1470);            // TCP 접속
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");    // TCP세션 상태 확인
        $rxlen = pid_ioctl($pid, "get rxlen");    // 수신 데이터 크기 확인
        $rlen = pid_recv($pid, $rdata, $rxlen); // 데이터 읽기
        echo "rlen = $rlen / ";                   // 읽은 데이터 크기 출력
        echo "rdata = $rdata\r\n";                // 읽은 데이터 출력
        if($rlen)
                $rdata = "";                      // 수신버퍼 초기화
}
while($state == TCP_CONNECTED);
pid_close($pid);
```

5.6.2 TCP 데이터 송신

pid_send함수를 이용해 송신한 데이터는 송신버퍼에 저장되었다가 네트워크로 전송됩니다.

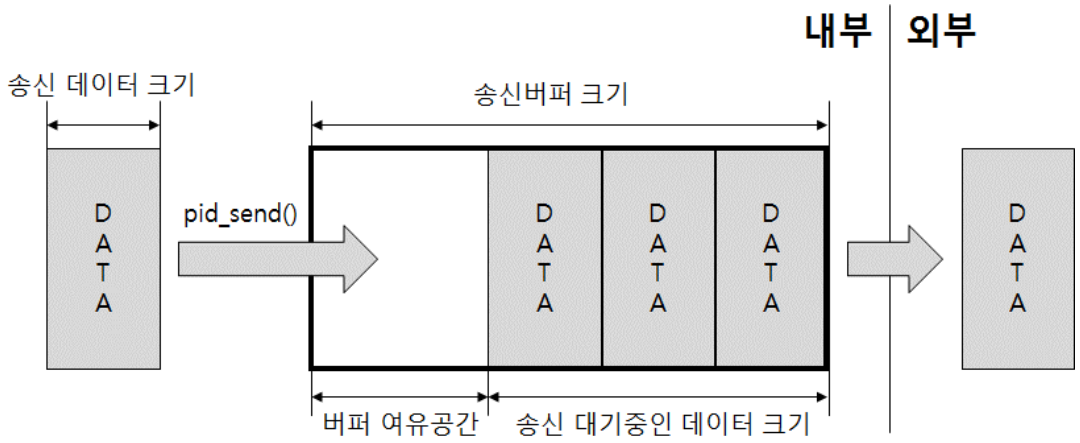


그림 5-3 TCP 데이터 송신

pid_send함수는 다음과 같이 사용합니다.

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

● 사용 예

이 예제는 송신버퍼의 여유공간을 확인하여 TCP로 데이터를 송신합니다.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/tcp0");                    // 0번 TCP 열기
pid_connect($pid, "10.1.0.2", 1470);              // TCP 접속
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");        // TCP세션 상태 확인
        $txfree = pid_ioctl($pid, "get txfree");      // 버퍼 여유공간 확인
        $tx_len = pid_send($pid, $sdata, $txfree); // 데이터 송신
        echo "tx len = $tx_len\r\n";                  // 송신 한 데이터 크기 출력
}
while($state == TCP_CONNECTED);
pid_close($pid);
```

위 코드에서 pid_send함수의 세 번째 인수는 송신할 데이터의 크기를 의미합니다. 송신 데이터크기가 송신버퍼의 여유공간보다 크면 데이터가 유실될 수 있습니다. 따라서 항상 버퍼 여유공간을 확인한 후 그 값 또는 그 이하로 쓰기 데이터 크기를 설정하는 것을 권장합니다.

6 UDP

UDP는 TCP와 달리 접속과정이 없어 데이터의 신뢰성이 보장되지 않지만, 프로토콜 헤더가 단순하고 속도가 빠르다는 특징이 있습니다.

6.1 사용 절차

일반적인 UDP 사용 절차는 다음과 같습니다.

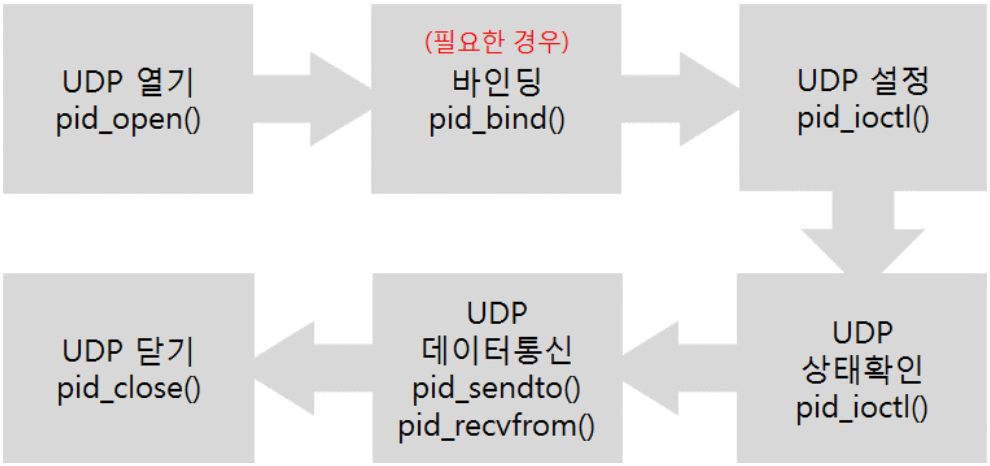


그림 6-1 UDP 사용 절차

☞ ***UDP데이터를 송신만 하거나 UDP설정을 하지 않는 경우에는 바인딩 과정을 생략할 수 있습니다.***

6.2 UDP 열기

UDP를 열기 위해서는 pid_open함수를 사용합니다.

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

☞ ***제품 별 UDP에 관한 자세한 내용은 부록을 참조하시기 바랍니다.***

6.3 UDP 바인딩

UDP데이터를 수신하거나 pid_ioctl함수로 수신 IP주소 또는 포트번호를 설정하기 위해서는 pid_bind함수를 사용하는 바인딩 과정이 필요합니다.

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

$addr은 바인딩 할 IP주소이고 $port는 바인딩 할 UDP포트번호 입니다. 바인딩 할 IP주소에 빈 문자열("")을 설정하면 현재 PHPoC에 설정 된 IP주소로 자동 설정 됩니다.

☞ ***바인딩 할 IP주소는 빈 문자열("") 이외의 값을 설정할 수 없습니다.***

- UDP 바인딩 예

```
$pid = pid_open("/mmap/udp0");      // 0번 UDP 열기
$port = 1470;                       // UDP 포트번호
pid_bind($pid, "", $port);          // 바인딩
```

6.4 UDP 설정

UDP를 사용하기 전에 수신 IP주소와 UDP포트번호를 미리 설정할 수 있습니다. 이 설정을 해 놓으면 pid_sendto함수로 데이터를 송신할 때 4번째와 5번째 인자를 생략할 수 있습니다.

설정을 위해서는 pid_ioctl함수의 set명령을 사용합니다.

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

ITEM은 설정 항목을, VALUE는 항목에 대한 값을 나타냅니다.

6.4.1 설정 가능한 UDP 항목

ITEM	VALUE	설명
dstaddr	예) 10.1.0.2	UDP통신 상대방의 IP주소
dstport	예) 1470	UDP통신 상대방의 포트번호

표 6-1 설정 가능한 UDP 항목

- UDP 설정 예

```
$pid = pid_open("/mmap/udp0");              // 0번 UDP 열기
pid_bind($pid, "", 1470);                   // 바인딩
pid_ioctl($pid, "set dstaddr 10.1.0.2");    // 통신 상대방 IP주소 설정
pid_ioctl($pid, "set dstport 1470");        // 통신 상대방 포트번호 설정
```

6.5 UDP 상태정보 확인

pid_ioctl함수의 get명령어로 UDP의 각종 상태를 확인 할 수 있습니다.

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

6.5.1 확인 가능한 UDP 상태정보

ITEM	설명	반환 값	반환 형식
srcaddr	UDP통신 - 송신 IP주소	예) 192.168.0.1	문자열
srcport	UDP통신 - 송신 포트번호	예) 1470	정수
dstaddr	UDP통신 - 수신 IP주소	예) 192.168.0.2	문자열
dstport	UDP통신 - 수신 포트번호	예) 1470	정수
rxlen	수신 데이터 량[Byte]	예) 200	정수

표 6-2 확인 가능한 UDP 상태 정보

6.5.2 수신 데이터 크기 확인

UDP수신 데이터 크기는 pid_ioctl함수의 "get rxlen"로 확인할 수 있습니다.

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

- 사용 예

이 예제는 PHPoC 장치가 UDP 수신 데이터 크기를 반복적으로 확인하다가 수신데이터가 있으면 데이터 크기를 콘솔로 출력하고 스크립트를 종료합니다.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                  // 0번 UDP 열기
pid_bind($pid, "", 1470);                       // 바인딩
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");     // 수신 데이터 크기 확인
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen);  // 데이터 수신
                echo "$rxlen bytes\r\n";          // 수신 데이터 크기 출력
        }
        usleep(100000);
}while($rxlen == 0);                            // 수신 데이터가 없는 동안 반복
pid_close($pid);
```

6.6 UDP 데이터 통신

6.6.1 데이터 수신

UDP 데이터를 수신하기 위해서는 pid_recvfrom함수를 사용합니다. UDP 수신버퍼는 2개이며, 다음과 같이 동작합니다.

☞ ***제품 별 UDP 수신버퍼의 크기는 부록을 참조하시기 바랍니다.***

- 네트워크로부터 데이터 수신

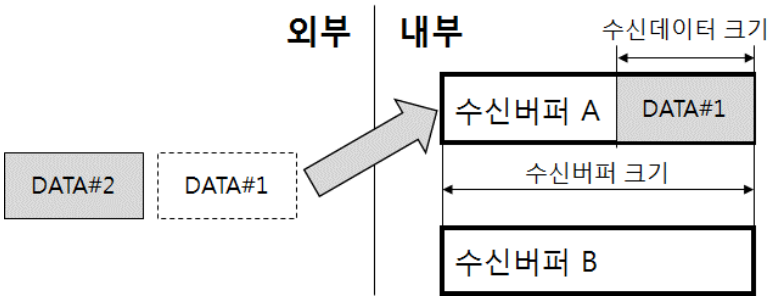


그림 6-2 네트워크로부터 데이터 수신

- 수신버퍼에 저장 된 데이터 읽기

  pid_recvfrom 함수를 호출하면 수신버퍼에 저장 된 데이터를 읽고, 버퍼를 비웁니다.

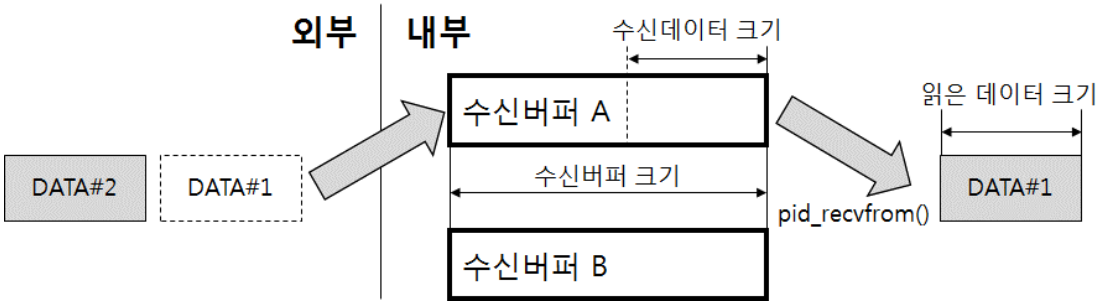


그림 6-3 수신버퍼에 저장 된 데이터 읽기

- 수신버퍼에 저장 된 데이터의 크기보다 작은 길이만큼 읽은 경우

  수신버퍼에 읽고 남은 데이터는 버퍼를 비움과 동시에 유실됩니다.

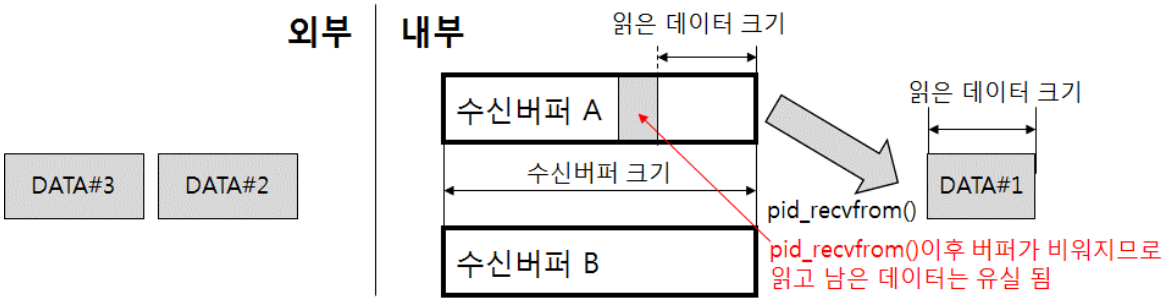


그림 6-4 수신버퍼에 읽고 남은 데이터가 있을 경우

● 두 개의 수신버퍼에 모두 데이터가 있는 경우

두 개의 수신버퍼에 모두 데이터가 있는 경우 수신버퍼가 비워지기 전까지는 다음에 들어오는 데이터가 모두 유실됩니다. 따라서 항상 수신버퍼를 확인하여 데이터가 있으면 바로 데이터를 읽도록 프로그래밍 하는 것을 권장합니다.

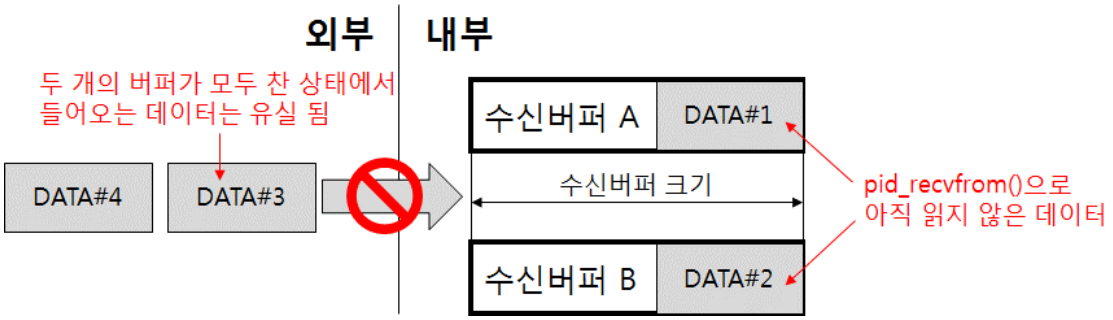


그림 6-5 두 개의 수신버퍼에 모두 데이터가 있는 경우

● 사용 예

이 예제는 UDP 수신 데이터 크기를 계속 확인하여 수신데이터가 있으면 데이터를 출력합니다.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                    // 0번 UDP 열기
pid_bind($pid, "", 1470);                         // 바인딩
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");    // 수신 데이터 크기 확인
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen);  // 데이터 수신
                echo "$rbuf\r\n";                   // 수신 데이터 출력
        }
        usleep(100000);
}while(1)                                         // 무한 루프
```

6.6.2 데이터 송신

UDP 데이터를 송신하기 위해서는 pid_sendto함수를 사용합니다.

● UDP 데이터 송신 예

```
$sdata = "01234567";
$pid = pid_open("/mmap/udp0");                    // 0번 UDP 열기
$slen = pid_sendto($pid, $sdata, 8, 0, "10.1.0.2", 1470); // 데이터 송신
echo "slen = $slen\r\n";                          // 송신한 데이터 크기 출력
pid_close($pid);
```

7 ST

PHPoC는 소프트웨어 타이머 디바이스 ST를 제공합니다.

7.1 사용 절차

일반적인 ST 사용 절차는 다음과 같습니다.

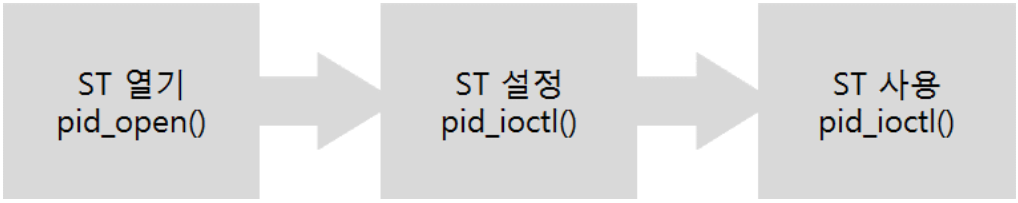


그림 7-1 ST 사용 절차

7.2 ST 열기

ST를 열기 위해서는 pid_open함수를 사용합니다.

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

☞ ***제품별 제공되는 ST에 대한 정보는 부록을 참조하시기 바랍니다.***

7.3 ST 설정 및 사용

ST를 설정하거나 사용하기 위해서는 pid_ioctl함수를 사용해야 합니다. ST는 사용 형태에 따라 다음 4가지 동작모드를 제공합니다.

모드	설명
프리모드	일반적인 카운터 모드
펄스출력모드	특정 핀으로 펄스신호를 출력하기 위한 모드
토글출력모드	특정 핀으로 토글신호를 출력하기 위한 모드
PWM출력모드	무한 펄스출력모드

표 7-1 ST 동작모드

7.3.1 공통 명령어

다음 명령어들은 모드와 상관없이 공통적으로 사용됩니다.

명령어	하위 명령어			설명
set	mode	free		모드 설정: 프리모드
		output	pulse	모드 설정: 펄스출력모드
			toggle	모드 설정: 토글출력모드
			pwm	모드 설정: PWM출력모드
	div	sec		단위 설정: 초
		ms		단위 설정: 밀리 초
		us		단위 설정: 마이크로 초
reset	-			초기화
get	state			상태 읽기
start	-			시작
stop	-			정지

표 7-2 공통 명령어

- 모드 설정

ST는 일반적인 카운터 모드인 프리모드와 신호를 출력하는 출력모드를 지원합니다. 출력모드에는 토글출력모드, 펄스출력모드 및 PWM출력모드가 있습니다. PWM출력모드는 출력횟수가 무한대인 펄스출력모드 입니다.
모드 설정 초기 값은 프리모드 이며, 각각의 모드 설정 방법은 다음과 같습니다.

구분	문법
프리모드	pid_ioctl($pid, "set mode free");
펄스출력모드	pid_ioctl($pid, "set mode output pulse");
토글출력모드	pid_ioctl($pid, "set mode output toggle");
PWM출력모드	pid_ioctl($pid, "set mode output pwm");

표 7-3 모드 설정

- 단위 설정

ST의 단위는 다음 세 가지로 설정할 수 있습니다. 초기 값은 밀리 초 입니다.

구분	문법
초	pid_ioctl($pid, "set div sec");
밀리 초	pid_ioctl($pid, "set div ms");
마이크로 초	pid_ioctl($pid, "set div us"");

표 7-4 단위 설정

- 초기화

"reset"명령어는 ST의 동작을 그 즉시 중단하고 초기화 합니다.

구분	문법
초기화	pid_ioctl($pid, "reset");

표 7-5 초기화

- 상태 읽기

"get state" 명령어는 ST의 상태를 읽는 명령어 입니다.

구분	문법
상태 읽기	pid_ioctl($pid, "get state");

표 7-6 상태 읽기

이 명령어에 의한 반환 값은 다음과 같습니다.

반환 값	상태
0	정지
1 ~ 5	동작 중

표 7-7 상태 읽기 반환 값

- 시작

ST를 시작시키기 위해서는 "start"명령을 사용합니다.

구분	문법
시작	pid_ioctl($pid, "start");

표 7-8 시작

- 정지

ST를 정지시키기 위해서는 "stop"명령을 사용합니다. 출력모드에서 ST를 정지시키면 출력 핀의 상태는 정지 시점의 상태를 유지합니다.

구분	문법
정지	pid_ioctl($pid, "stop");

표 7-9 정지

7.3.2 프리모드

프리모드는 ST를 일반적인 카운터로 동작시키는 모드 입니다.

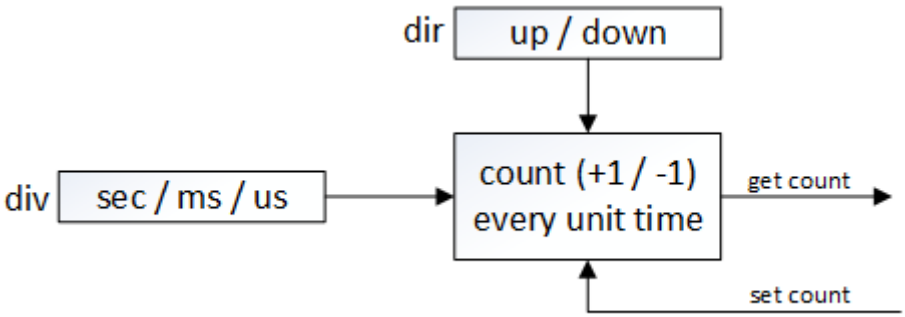


그림 7-2 프리모드 블록도

프리모드에서 사용 가능한 pid_ioctl함수의 명령어는 다음과 같습니다.

명령어	하위 명령어		설명
set	mode	free	모드 설정: 프리모드
	div	sec	단위 설정: 초
		ms	단위 설정: 밀리 초
		us	단위 설정: 마이크로 초
	dir	up	업카운터로 설정
		down	다운카운터로 설정
	count	[T]	다운카운터 초기 값을 [T]로 설정
reset	-		초기화
get	count		카운트 값 읽기
	state		상태 읽기
start	-		시작
stop	-		정지

표 7-10 프리모드 명령어

- 카운터 종류 설정

  ST의 카운터 종류는 업카운터 또는 다운카운터로 설정할 수 있습니다. 초기 값은 업카운터 입니다.

구분	문법
업카운터(UP)	pid_ioctl($pid, "set dir up");
다운카운터(DOWN)	pid_ioctl($pid, "set dir down");

표 7-11 카운터 종류 설정

- 카운트 값 설정

  프리모드에서는 다운카운터로 설정했을 때 카운터의 초기 값을 설정합니다. 카운트 설정 방법은 다음과 같습니다.

구분	문법
프리모드	pid_ioctl($pid, "set count T");

표 7-12 카운트 설정

  만약 업카운터일 때 T를 설정하면 해당 값은 반영되지 않습니다. 즉, 업카운터일 때 타이머 초기 값은 항상 0입니다. 다운카운터에서 설정 가능한 T의 범위는 다음과 같습니다.

구분	T 설정 범위
프리모드	0 ~ (2의 64제곱 - 1)

표 7-13 설정 가능한 카운트 값의 범위

- 카운트 값 읽기

  "get count"명령어는 ST의 현재 카운트 값을 읽는 명령어 입니다.

구분	문법
프리모드	pid_ioctl($pid, "get count");

표 7-14 카운트 읽기

7.3.3 프리모드 사용 예

프리모드에서의 ST값은 pid_ioctl함수의 "get count"명령으로 읽을 수 있으며 이 값은 타이머가 가동 된 시점부터 값을 읽는 시점까지 소요된 시간을 의미합니다.

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

- 업카운터

  이 예제는 ST를 업카운터로 설정하고 약 1초마다 ST값을 읽어와 출력합니다.

```
$pid = pid_open("/mmap/st0");                  // 0번 ST 열기
pid_ioctl($pid, "set mode free");              // 프리모드 설정
pid_ioctl($pid, "set div sec");                // 단위 설정: 초
pid_ioctl($pid, "set dir up");                 // 업카운터 설정
pid_ioctl($pid, "start");                      // ST 시작
for($i=0; $i<10; $i++)
{
        $value = pid_ioctl($pid, "get count"); // 카운트 값 읽기
        echo "$value\r\n";                     // 카운트 값 출력
        sleep(1);
}
pid_close($pid);
```

- 다운카운터

  이 예제는 ST를 다운카운터로 설정하고 초기 값을 10초로 하여 약 1초마다 ST값을 읽어와 출력합니다.

```
$pid = pid_open("/mmap/st0");                  // 0번 ST열기
pid_ioctl($pid, "set mode free");              // 프리모드 설정
pid_ioctl($pid, "set div sec");                // 단위 설정: 초
pid_ioctl($pid, "set dir down");               // 다운카운터 설정
pid_ioctl($pid, "set count 10");               // 카운트 값을 10으로 초기화
pid_ioctl($pid, "start");                      // ST 시작
for($i = 0; $i < 10; $i++)
{
        $value = pid_ioctl($pid, "get count"); // 카운트 값 읽기
        echo "$value\r\n";                     // 카운트 값 출력
        sleep(1);
}
pid_close($pid);
```

7.3.4 토글출력모드

토글출력모드는 지정 된 핀으로 토글신호를 출력하는 모드입니다.

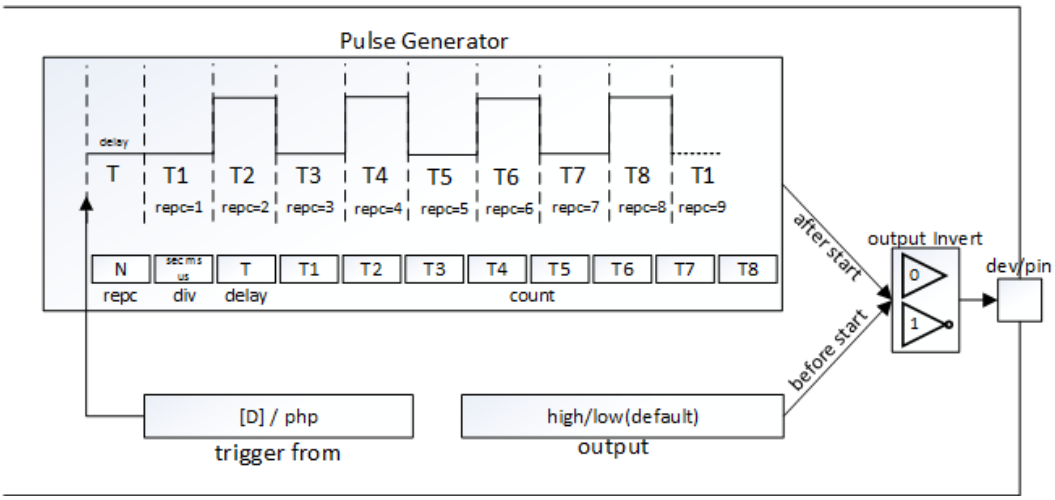


그림 7-3 토글출력모드 블록도

토글출력모드에서 사용 가능한 명령어는 다음과 같습니다.

명령어	하위 명령어				설명
set	mode	output	toggle		모드 설정: 토글출력모드
	div	sec			단위 설정: 초
		ms			단위 설정: 밀리 초
		us			단위 설정: 마이크로 초
	output	low			LOW 출력
		high			HIGH 출력
		dev	io3/io4	#pin	출력 핀 설정: io3/io4의 #pin
		invert	0		정상(비 반전)출력
			1		반전 출력
	count	[T1] … [T8]			출력 타이밍 설정
	delay	[D]			출력 지연시간 설정
	repc	[N]			출력 횟수 설정
	trigger	from	st#		트리거 대상 설정: st0 ~ st7
			php		트리거 대상 설정: 없음
reset	-				초기화
get	state				상태 읽기
	repc				남은 출력 횟수 읽기
start	-				시작
stop	-				정지

표 7-15 토글출력모드 명령어

- 출력 설정

  ST의 출력 설정은 다음과 같은 항목들이 있습니다.

구분	문법
출력 핀	pid_ioctl($pid, "set output dev io3 0"); // io3의 0번 핀
HIGH출력	pid_ioctl($pid, "set output high");
LOW출력	pid_ioctl($pid, "set output low");

반전출력	pid_ioctl($pid, "set output invert 0"); // 정상출력
	pid_ioctl($pid, "set output invert 1"); // 반전출력

표 7-16 출력 설정

  모든 출력 명령은 명령어 실행과 동시에 출력 핀에 반영됩니다.

- 지연시간 설정

  출력모드에서는 출력 전 지연시간을 줄 수 있습니다. 지연시간 설정 단위는 "set div"명령에 의한 단위가 사용됩니다. 지연시간 설정 방법은 다음과 같습니다.

구분	문법
지연시간	pid_ioctl($pid, "set delay D");

표 7-17 지연시간 설정

- 반복횟수 설정

  출력모드에서는 출력 신호의 반복횟수를 설정할 수 있습니다. 설정 가능한 N의 범위는 0 ~ 1,000,000,000(10억) 입니다. 기본 값은 0이며 0은 최대 반복횟수인 10억을 의미합니다.

구분	문법
반복횟수	pid_ioctl($pid, "set repc N");

표 7-18 반복횟수 설정

- 카운트 값 설정

  카운트 값 설정은 출력 타이밍을 조정하기 위해 사용합니다. 토글출력모드에서는 카운트 값을 최소 1개에서 최대 8개까지 설정할 수 있습니다. 설정 방법은 다음과 같습니다.

구분	문법
토글출력모드	pid_ioctl($pid, "set count T1 T2 … T8");

표 7-19 카운트 값 설정

토글출력모드에서 가능한 카운트 값의 범위는 다음과 같습니다.

단위	설정 가능 범위(10㎲ ~ 30분)
마이크로 초	10 ~ 1,800,000,000
밀리 초	1 ~ 1,800,000
초	1 ~ 1,800

표 7-20 설정 가능한 카운트 값의 범위

지연시간이 D이고 카운트 값을 하나(T1)만 설정할 때의 타이밍은 다음과 같습니다.

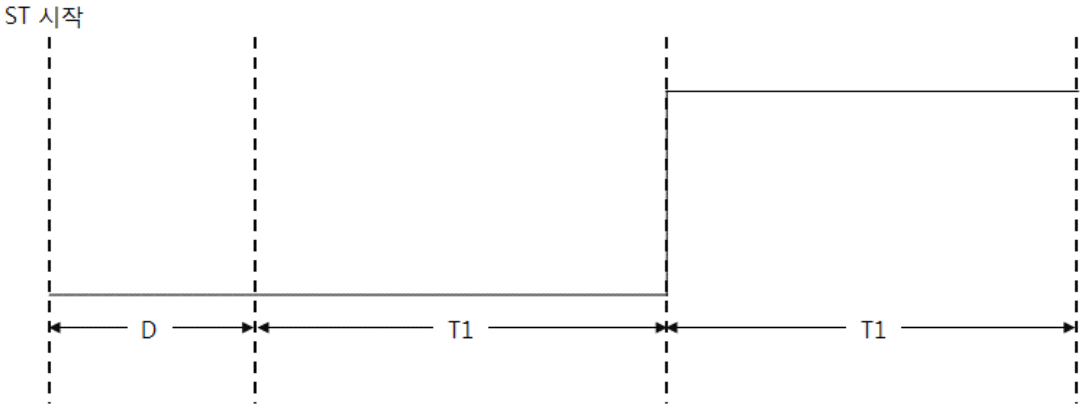


그림 7-4 토글출력모드에서 T1 타이밍(LOW -> HIGH)

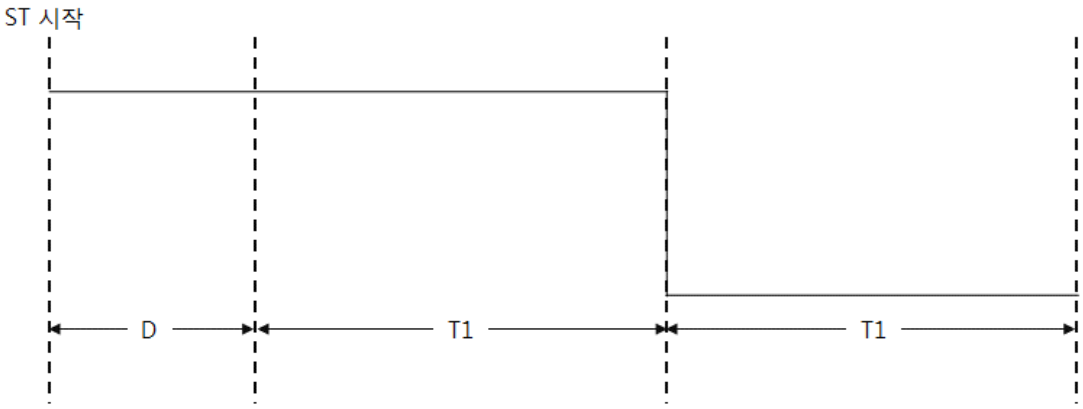


그림 7-5 토글출력모드에서 T1 타이밍(HIGH -> LOW)

토글출력모드에서 카운트 값을 2개 이상 설정하면 이 값들은 출력 시점마다 순서대로 사용됩니다. 만약 반복횟수가 설정 한 카운트 값의 수 보다 많으면 다시 첫 번째 값부터 순서대로 사용됩니다. 반복횟수가 4이고 지연시간이 D일 때 3개의 값(T1 ~ T3)을 설정하는 경우의 타이밍은 다음과 같습니다.

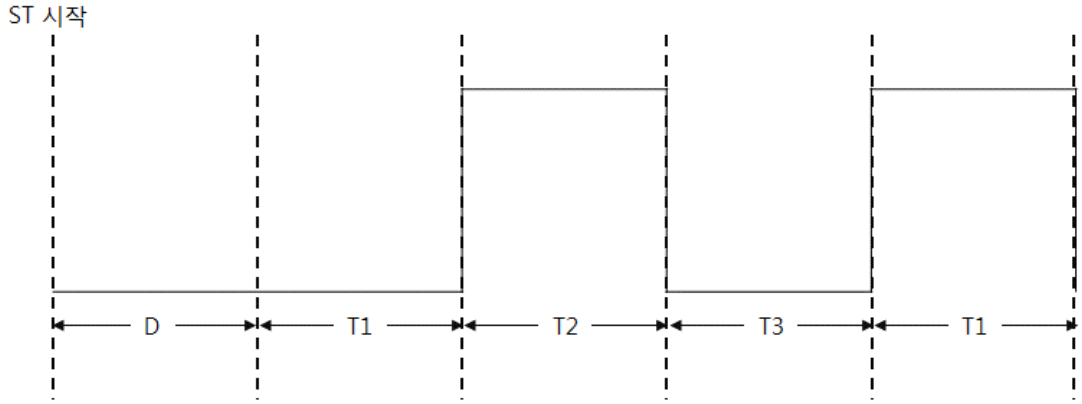


그림 7-6 토글출력모드에서 T1, T2 및 T3 타이밍

- 트리거 설정

트리거는 ST를 출력모드로 사용할 때 시작 시점을 동기화 하기 위해 사용합니다. 이 때 ST의 시작 시점은 트리거 대상과 동기화 되며 트리거 대상은 또 다른 ST만 설정이 가능합니다. ST의 트리거 설정 방법은 다음과 같습니다.

구분	문법
ST(st0/1...)	pid_ioctl($pid, "set trigger from st0");
php	pid_ioctl($pid, "set trigger from php");

표 7-21 트리거 설정

ST의 트리거 설정 기본 값은 "대상 없음(php)" 입니다.

- 남은 출력 횟수 읽기

"get repc"명령어는 ST의 남은 출력 횟수를 확인하는 명령어 입니다.

구분	문법
남은 출력 횟수	pid_ioctl($pid, "get repc");

표 7-22 남은 출력 횟수 읽기

7.3.5 토글출력모드 사용 예

토글출력모드는 ST의 출력신호를 반전시키는 모드 입니다.

- 토글출력모드 사용

```
$pid = pid_open("/mmap/st0");                    // 0번 ST 열기
pid_ioctl($pid, "set div sec");                  // 단위 설정: 초
pid_ioctl($pid, "set mode output toggle");       // 토글출력모드 설정
pid_ioctl($pid, "set output dev io3 0");         // 출력 핀 설정: io3의 0번
pid_ioctl($pid, "set repc 1");                   // 출력 횟수 설정: 1
pid_ioctl($pid, "set count 1");                  // 카운트 값 설정: 1
pid_ioctl($pid, "start");                        // ST 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

toggle모드에서의 "set count"는 ST의 시작시점부터 토글신호를 내보내는 시점까지의 시간을 의미합니다. 위 예제를 실행했을 때 ST의 출력 결과는 다음과 같습니다.

그림 7-7 토글출력모드 예

● 반복적 토글출력모드

```
$pid = pid_open("/mmap/st0");                  // 0번 ST 열기
pid_ioctl($pid, "set div sec");                // 단위 설정: 초
pid_ioctl($pid, "set mode output toggle");     // 토글출력모드 설정
pid_ioctl($pid, "set output dev io3 0");       // 출력 핀 설정: io3의 0번 핀
pid_ioctl($pid, "set repc 3");                 // 출력 횟수 설정: 3
pid_ioctl($pid, "set count 1 2 1");            // 카운트 값 설정: 1, 2, 1
pid_ioctl($pid, "start");                      // ST 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

위 예제에서는"set repc"명령어로 토글 출력 횟수를 3회로 설정하고 토글 출력 타이밍을 위해 "set count"로 T1, T2 및 T3를 각각 1, 2 그리고 1초로 설정했습니다. 위 예제를 실행했을 때 ST출력 결과는 다음과 같습니다.

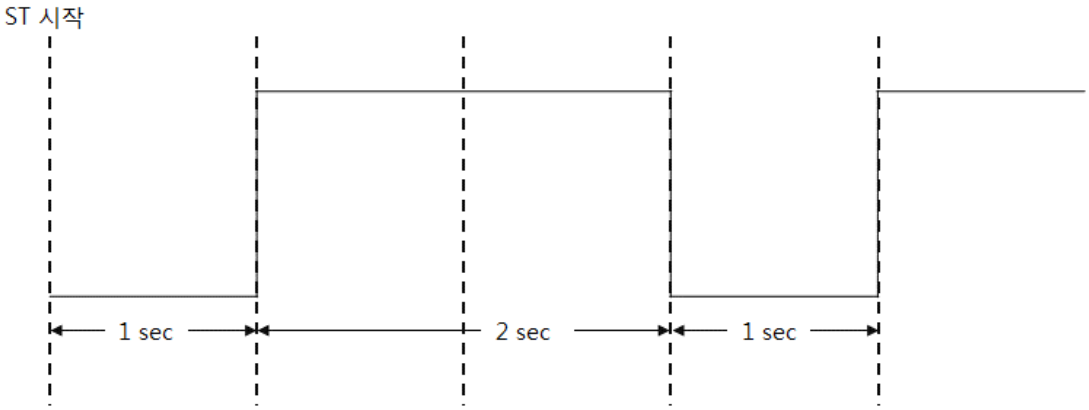


그림 7-8 반복적 토글출력모드

7.3.6 펄스출력모드

펄스출력모드는 구형파(펄스)를 출력하는 모드입니다.

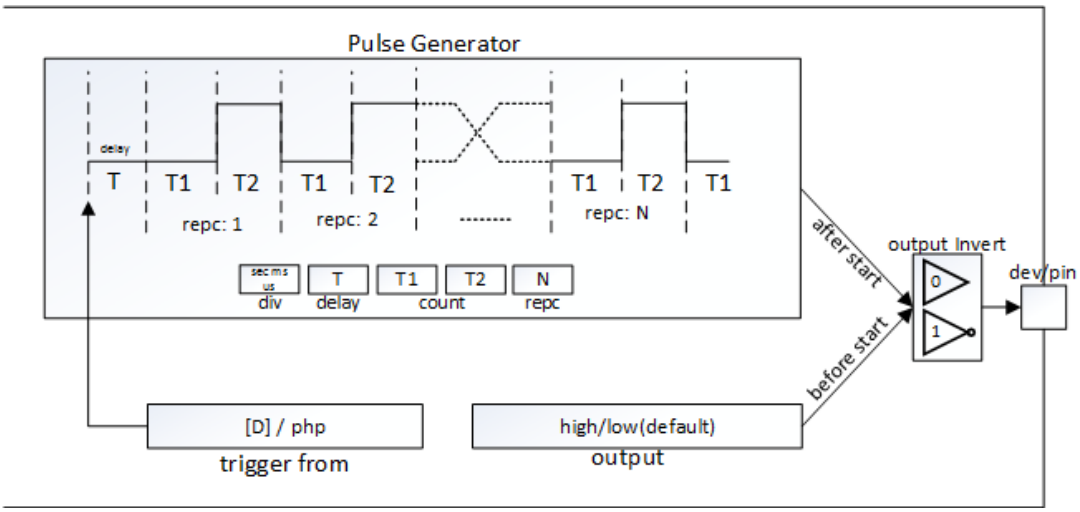


그림 7-9 펄스출력모드 블록도

펄스출력모드에서 사용 가능한 명령어는 다음과 같습니다.

명령어	하위 명령어				설명
set	mode	output	pulse		모드 설정: 펄스출력모드
	div	sec			단위 설정: 초
		ms			단위 설정: 밀리 초
		us			단위 설정: 마이크로 초
	output	low			LOW 출력
		high			HIGH 출력
		dev	io3/4	#pin	출력 핀 설정: io3/4의 #pin
		invert	0		정상(비 반전)출력
			1		반전 출력
	count	[T1] [T2]			출력 타이밍 설정
	delay	[D]			출력 지연시간 설정
	repc	[N]			출력 횟수 설정
	trigger	from	st#		트리거 대상 설정: st0 ~ st7
			php		트리거 대상 설정: 없음
reset	-				초기화
get	state				상태 읽기
	repc				남은 출력 횟수 읽기
start	-				시작
stop	-				정지

표 7-23 펄스출력모드 명령어

- 출력 설정

  ST의 출력 설정은 출력모드에서 사용이 가능하며 다음과 같은 항목들이 있습니다.

구분	문법
출력 핀	pid_ioctl($pid, "set output dev io3 0"); // io3의 0번 핀
HIGH출력	pid_ioctl($pid, "set output high");
LOW출력	pid_ioctl($pid, "set output low");

반전출력	pid_ioctl($pid, "set output invert 1"); // 반전출력
	pid_ioctl($pid, "set output invert 0"); // 정상출력

표 7-24 출력 설정

  모든 출력 명령은 명령어 실행과 동시에 출력 핀에 반영됩니다.

- 지연시간 설정

  출력모드에서는 출력 전 지연시간을 줄 수 있습니다. 지연시간 설정 단위는 "set div"명령에 의한 단위가 사용됩니다. 지연시간 설정 방법은 다음과 같습니다.

구분	문법
지연시간	pid_ioctl($pid, "set delay D");

표 7-25 지연시간 설정

- 반복횟수 설정

  출력모드에서는 출력 신호의 반복횟수를 설정할 수 있습니다. 설정 가능한 N의 범위는 0 ~ 10억 입니다. 기본 값은 0이며 0은 최대 반복횟수인 10억을 의미합니다.

구분	문법
반복횟수	pid_ioctl($pid, "set repc N");

표 7-26 반복횟수 설정

● 카운트 값 설정

카운트 값은 출력 타이밍을 조정하기 위해 설정합니다. 펄스출력모드에서 카운트 값 설정 방법은 다음과 같습니다.

구분	문법
카운트 설정	pid_ioctl($pid, "set count T1 T2");

표 7-27 카운트 값 설정

펄스출력모드에서 설정 가능한 카운트 값의 범위는 다음과 같습니다.

단위	카운트 값 설정 가능 범위(0 ~ 30분)
마이크로 초	0, 10 ~ 1,800,000,000
밀리 초	0 ~ 1,800,000
초	0 ~ 1,800

표 7-28 설정 가능한 카운트 값의 범위

펄스출력모드에서는 두 가지 카운트 값(T1과 T2)의 설정이 필요합니다. 펄스출력모드에서 반복횟수가 2이고 지연시간이 D일때 T1과 T2의 타이밍은 다음과 같습니다. 여기서 D는 지연시간을 의미합니다.

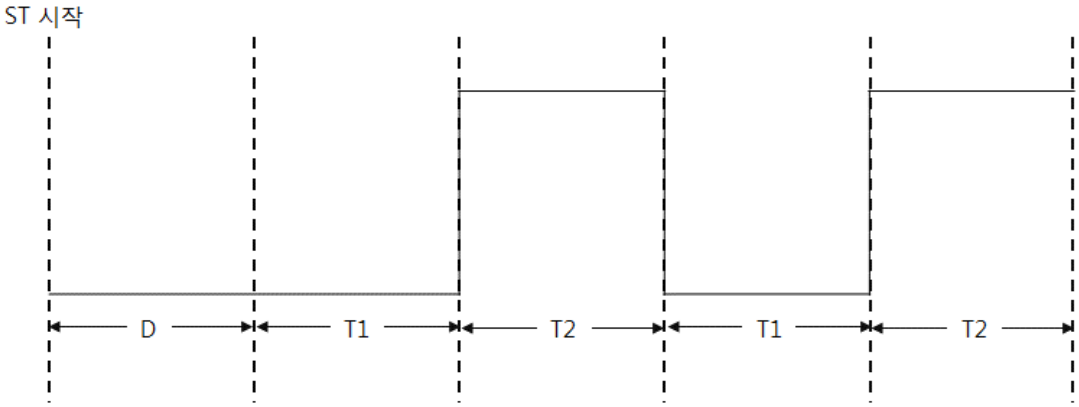


그림 7-10 펄스출력모드 T1, T2 타이밍

● 반복횟수 읽기

"get repc"명령어는 ST의 남은 출력 횟수를 확인하는 명령어 입니다.

구분	문법
남은 출력 횟수	pid_ioctl($pid, "get repc");

표 7-29 반복횟수 읽기

7.3.7 펄스출력모드 사용 예

- 펄스출력모드 사용(HIGH펄스)

```
$pid = pid_open("/mmap/st0");                    // 0번 ST 열기
pid_ioctl($pid, "set div sec");                  // 단위 설정: 초
pid_ioctl($pid, "set mode output pulse");        // 펄스출력모드 설정
pid_ioctl($pid, "set output dev io3 0");         // 출력 핀 설정: io3의 0번
pid_ioctl($pid, "set count 1 2");                // 카운트 값 설정: 1, 2
pid_ioctl($pid, "set repc 1");                   // 출력 횟수 설정: 1
pid_ioctl($pid, "start");                        // ST 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

펄스출력모드의 출력은 기본적으로 HIGH펄스입니다. HIGH상태를 유지하는 시간은 설정 단위와 "set count"로 설정하는 T1과 T2에 의해 결정됩니다. 위 예제를 실행했을 때 ST출력 결과는 다음과 같습니다.

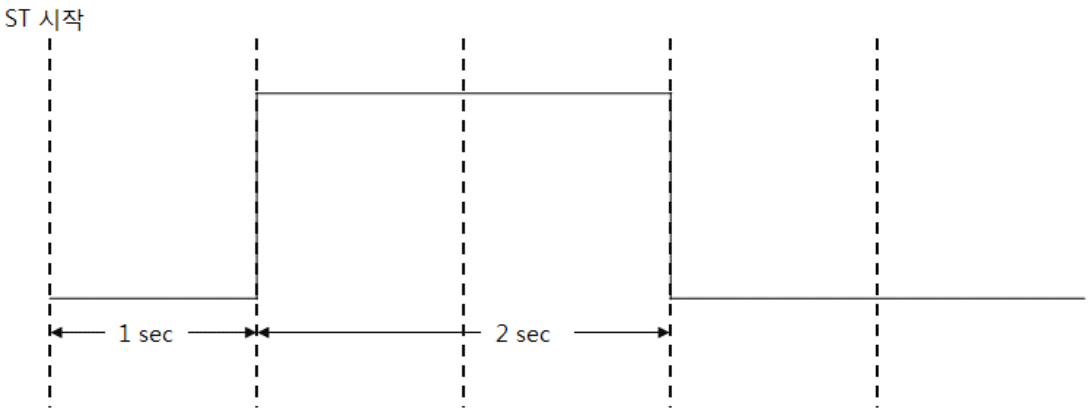


그림 7-11 펄스출력모드(HIGH펄스)

● 펄스출력모드 사용(LOW펄스)

```
$pid = pid_open("/mmap/st0");                  // 0번 ST 열기
pid_ioctl($pid, "set div sec");                // 단위 설정: 초
pid_ioctl($pid, "set mode output pulse");      // 펄스출력모드 설정
pid_ioctl($pid, "set output dev io3 0");       // 출력 핀 설정: io3의 0번
pid_ioctl($pid, "set count 1 2");              // 카운트 값 설정: 1, 2
pid_ioctl($pid, "set output invert 1");        // 출력 반전
pid_ioctl($pid, "set repc 1");                 // 출력 횟수 설정: 1
pid_ioctl($pid, "start");                      // ST 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
```

"set output invert 1"명령을 수행하면 이후의 출력이 반전됩니다. 펄스출력도 반전이 되므로 위 예제를 실행했을 때 ST의 출력 결과는 다음과 같습니다.

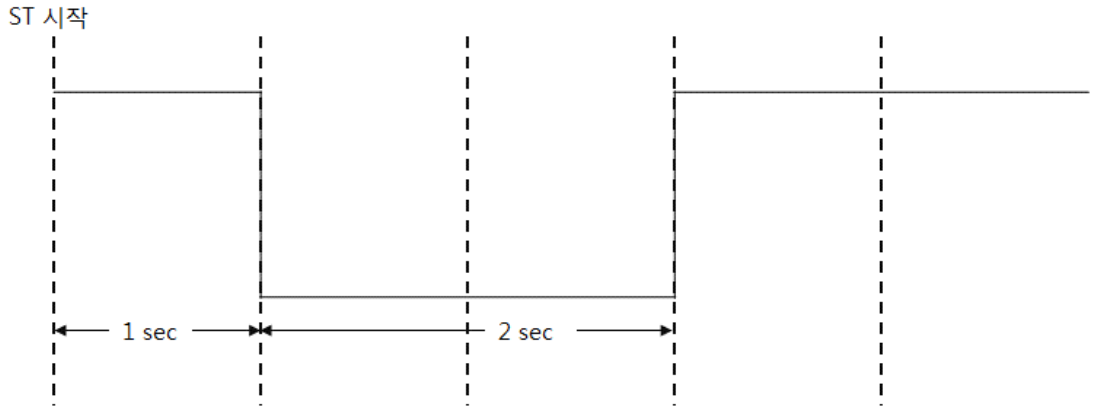


그림 7-12 펄스출력모드(LOW펄스)

7.3.8 PWM 출력모드

PWM출력모드는 반복횟수가 무한한 펄스출력모드입니다. 따라서 반복횟수와 카운트 설정을 제외하고 모든 사용 방법은 펄스출력모드와 동일합니다.

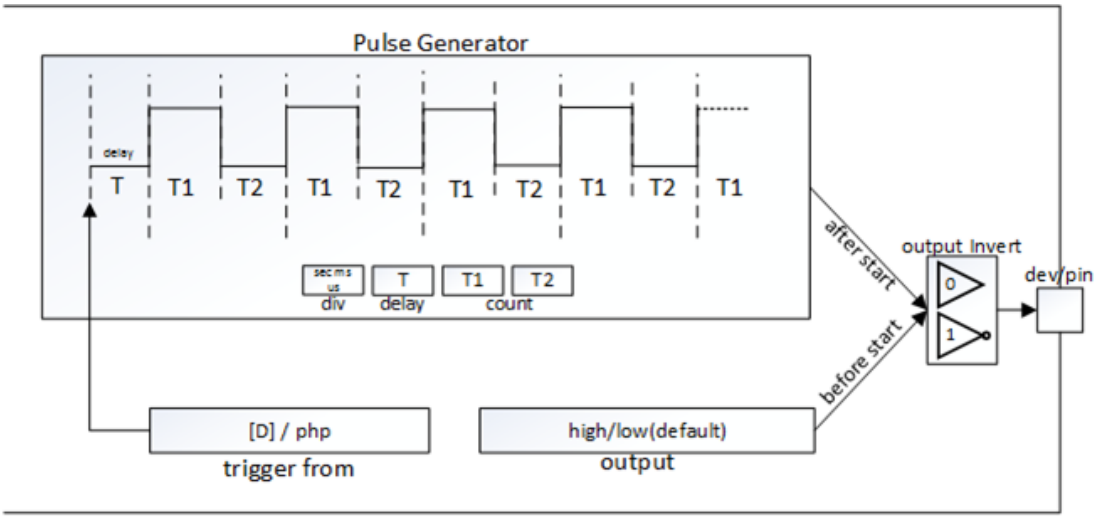


그림 7-13 PWM출력모드 블록도

PWM출력모드에서 사용 가능한 명령어는 다음과 같습니다.

명령어	하위 명령어				설명
set	mode	output	pwm		모드 설정: PWM출력모드
	div	sec			단위 설정: 초
		ms			단위 설정: 밀리 초
		us			단위 설정: 마이크로 초
	output	low			LOW 출력
		high			HIGH 출력
		dev	io3/4	#pin	출력 핀 설정: io3/4의 #pin
		invert	0		정상(비 반전)출력
			1		반전 출력
	count	[T1] [T2]			출력 타이밍 설정
	delay	[D]			출력 지연시간 설정
	trigger	from	st#		트리거 대상 설정: st0 ~ st7
			php		트리거 대상 설정: 없음
reset	-				초기화
get	state				상태 읽기
start	-				시작
stop	-				정지

표 7-30 PWM출력모드 명령어

● 카운트 값 설정

카운트 값은 출력 타이밍을 조정하기 위해 설정합니다. PWM출력모드에서는 두 개의 카운트 값 설정이 필요합니다. 카운트 값 설정 방법은 다음과 같습니다.

구분	문법
카운트 값 설정	pid_ioctl($pid, "set count T1 T2");

표 7-31 카운트 값 설정

PWM출력모드에서 설정 가능한 카운트 값의 범위는 다음과 같습니다.

단위	설정 가능한 카운트 값 (0 ~ 30분)
마이크로 초	0, 10 ~ 1,800,000,000
밀리 초	0 ~ 1,800,000
초	0 ~ 1,800

표 7-32 설정 가능한 카운트 값의 범위

PWM출력모드에서 지연시간이 D일때 T1과 T2의 타이밍은 다음과 같습니다. 여기서 D는 지연시간을 의미합니다.

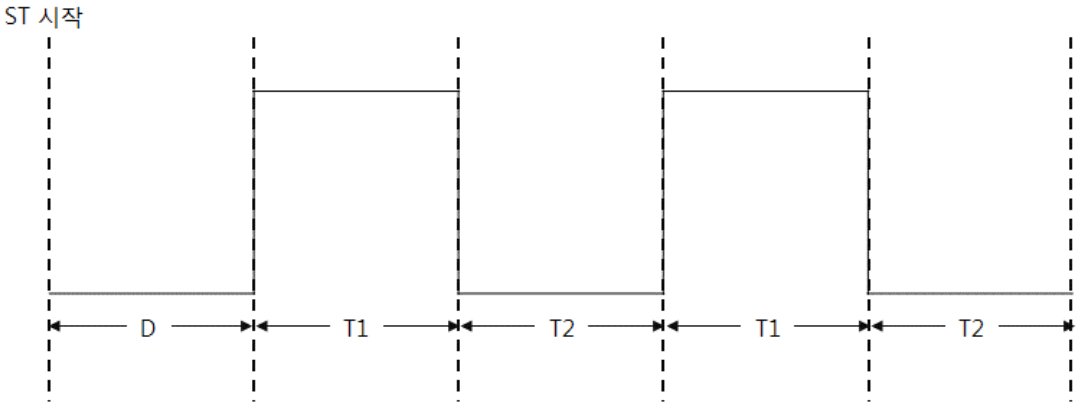


그림 7-14 PWM출력모드

7.3.9 PWM 출력모드 사용 예

- PWM출력모드 사용 예

```
$pid = pid_open("/mmap/st0");                  // 0번 ST 열기
pid_ioctl($pid, "set div sec");                // 단위 설정: 초
pid_ioctl($pid, "set mode output pwm");        // PWM출력모드 설정
pid_ioctl($pid, "set output dev io3 0");       // 출력 핀 설정: io3의 0번
pid_ioctl($pid, "set count 1 1");              // 카운트 값 설정: 1, 1
pid_ioctl($pid, "start");                      // ST 시작
sleep(10);
pid_ioctl($pid, "stop");                       // ST 정지
pid_close($pid);
```

위 예제를 실행했을 때 ST의 출력 결과는 다음과 같습니다.

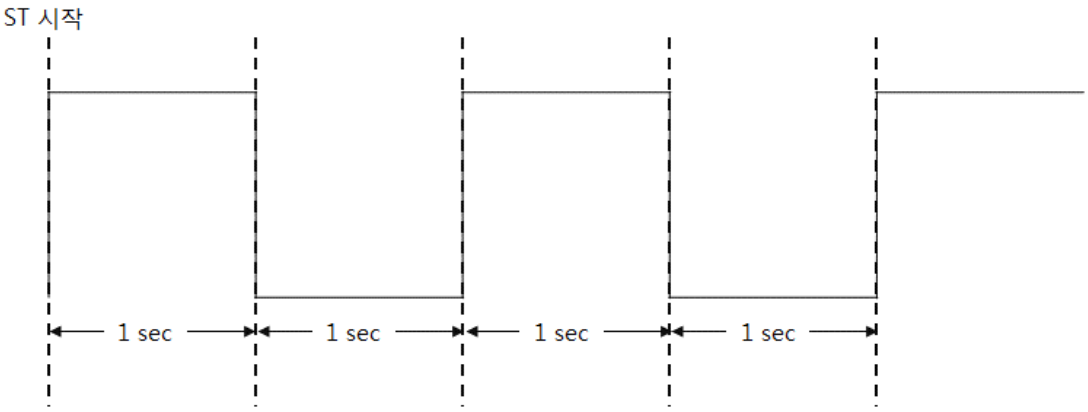


그림 7-15 PWM출력모드 예

7.3.10 트리거

트리거 명령은 ST의 출력 시점을 또 다른 ST와 동기화시키고자 할 때 사용합니다. 아래 예제는 ST1의 출력 시점을 ST0와 동기화시키는 예 입니다.

- 트리거 사용 예

```
$pid0 = pid_open("/mmap/st0");                  // 0번 ST 열기
pid_ioctl($pid0, "set div sec");                // 단위 설정: 초
pid_ioctl($pid0, "set mode output pulse");      // 펄스출력모드 설정
pid_ioctl($pid0, "set count 1 1");              // 카운트 값 설정: 1, 1
pid_ioctl($pid0, "set repc 2");                 // 출력 횟수 설정: 2
pid_ioctl($pid0, "set output dev io3 0");       // 출력 핀 설정: io3의 0번

$pid1 = pid_open("/mmap/st1");                  // 1번 ST 열기
pid_ioctl($pid1, "set div sec");                // 단위 설정: 초
pid_ioctl($pid1, "set mode output pulse");      // 펄스출력모드 설정
pid_ioctl($pid1, "set trigger from st0");       // 트리거 대상 ST 지정: st0
pid_ioctl($pid1, "set count 1 1");              // 카운트 값 설정: 1, 1
pid_ioctl($pid1, "set repc 2");                 // 출력 횟수 설정: 2
pid_ioctl($pid1, "set output dev io3 1");       // 출력 핀 설정: io3의 1번

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

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

위 예제에서 보는 바와 같이, 출력을 동기화시킬 ST는 트리거의 대상이 되는 ST보다 반드시 먼저 가동되어야 합니다.

- ST의 트리거 오차

ST는 트리거 오차가 존재합니다. ST의 트리거 오차 범위는 다음과 같습니다.

구분	오차 범위
2개의 타이머 사용 시	약 1μs
8개의 타이머 사용 시	약 4μs

표 7-33 트리거 오차

8 부록: 디바이스 관련 함수

PHPoC는 다음과 같은 디바이스 사용 관련 내부함수를 제공합니다.

함수 이름	사용 법
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]);

표 8-1 PHPoC의 디바이스 관련 내부 함수

☞ ***각 함수에 대한 자세한 내용은 PHPoC Internal Functions문서를 참조하시기 바랍니다.***

9 부록: 디바이스 정보

9.1 제품 별 디바이스 개수

구분		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

표 9-1 제품 별 디바이스 수

9.2 제품 별 디바이스 파일 경로

9.2.1 UART

구분	파일 경로
PBH-101, PBH-204	/mmap/uart0
PBH-104	/mmap/uart0
	/mmap/uart1
	/mmap/uart2
	/mmap/uart3

표 9-2 UART

9.2.2 NET

구분	파일 경로	비고
PBH-101, PBH-104, PBH-204	/mmap/net0	유선랜
	/mmap/net1	무선랜

표 9-3 NET

9.2.3 TCP

구분	파일 경로
PBH-101, PBH-104, PBH-204	/mmap/tcp0
	/mmap/tcp1
	/mmap/tcp2
	/mmap/tcp3
	/mmap/tcp4

표 9-4 TCP

9.2.4 UDP

구분	파일 경로
PBH-101, PBH-104, PBH-204	/mmap/udp0
	/mmap/udp1
	/mmap/udp2
	/mmap/udp3
	/mmap/udp4

표 9-5 UDP

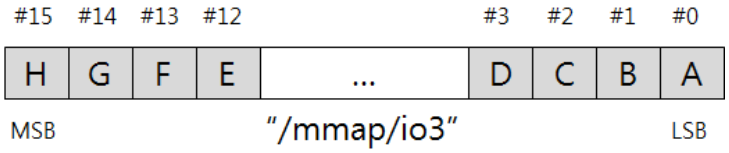
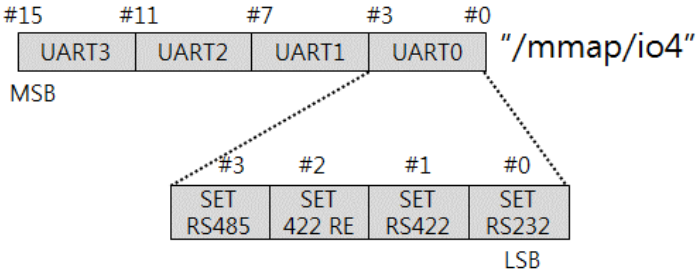
9.2.5 I/O

- PBH-101

<table>
<tr><th colspan="2">구분</th><th>파일 경로 및 맵핑 정보</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
(시리얼
통신방식)</td><td>/mmap/io4
#3 #2 #1 #0
... SET RS485 SET 422 RE SET RS422 SET RS232
MSB "/mmap/io4" LSB
● 통신방식 별 설정 예
<table><tr><th>구분</th><th>값</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>

표 9-6 PBH-101 디지털 I/O

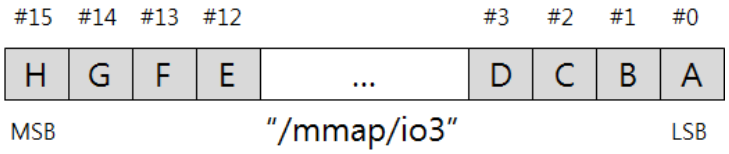
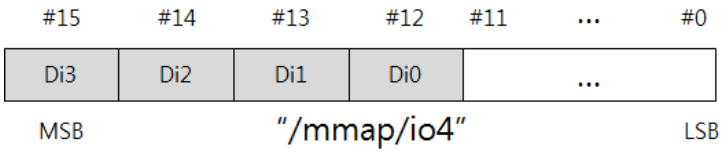
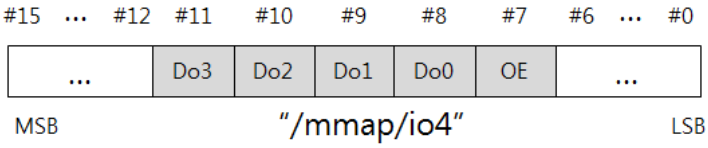
● PBH-104

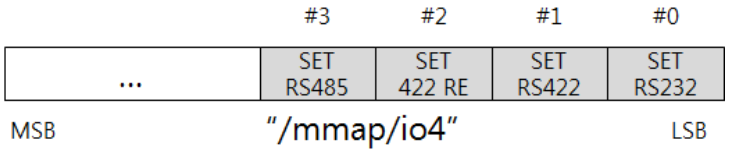
<table>
<tr><th colspan="2">구분</th><th>파일 경로 및 맵핑 정보</th></tr>
<tr><td rowspan="2">PBH-104</td><td>LED</td><td>/mmap/io3
 
</td></tr>
<tr><td>UART Mode
(시리얼
통신방식)</td><td>/mmap/io4
 

● 통신방식 별 설정 예
<table>
<tr><th>구분</th><th>값</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>
● 포트 별 설정 예
<table>
<tr><th>구분</th><th>일괄설정</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>

표 9-7 PBH-104 디지털 I/O

● PBH-204

<table>
<tr><th colspan="2">구분</th><th>파일 경로 및 맵핑 정보</th></tr>
<tr><td rowspan="4">PBH-204</td><td>LED</td><td>/mmap/io3
 
</td></tr>
<tr><td>Digital Input (photo-coupler)</td><td>/mmap/io4
 
</td></tr>
<tr><td>Digital Output (Relay)</td><td>/mmap/io4
 

※ OE: 출력 릴레이를 Enable / Disable 하기 위한 비트
- Enable: LOW(0), Disable: HIGH(1)</td></tr>
<tr><td>UART Mode</td><td>/mmap/io4
 

● 통신방식 별 설정 예
<table>
<tr><th>구분</th><th>값</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>

표 9-8 PBH-204 디지털 I/O

9.2.6 ST

구분	파일 경로
PBH-101, PBH-104, PBH-204	/mmap/st0
	/mmap/st1
	/mmap/st2
	/mmap/st3
	/mmap/st4
	/mmap/st5
	/mmap/st6
	/mmap/st7

표 9-9 ST

9.2.7 ENV 및 사용자메모리

구분		파일 경로	크기(바이트)
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

표 9-10 ENV 및 사용자메모리

10 부록: 펌웨어 사양 및 제한사항

10.1 펌웨어별 적용 제품

펌웨어	적용 제품
P20	PBH-101, PBH-104, PBH-204

표 10-1 펌웨어별 적용 제품

10.2 펌웨어 사양

항목	사양(p20)	설명
ENVS	1,536	시스템 ENV 크기, byte
ENVU	1,536	사용자 ENV 크기, byte
WLAN	1	무선랜
EMAC	1	이더넷
UART	4	UART 개수
FLOAT	지원	부동소수점 수
SSL	지원	SSL보안통신
PHP_MAX_NAME_SPACE	16	네임스페이스 수
PHP_NAME_LEN	32	사용자 식별자 길이, byte
PHP_MAX_USER_DEF_NAME	480	사용자 식별자 개수
PHP_LLSTR_BLK_SIZE	64	문자열블록 크기, byte
PHP_MAX_LLSTR_BLK	192	문자열블록 개수
string buffer size	12K	문자열버퍼 크기, byte
PHP_MAX_STRING_LEN	1,536	문자열변수 최대 길이, byte
PHP_INT_MAX	약 $9.2*10^{18}$	정수 값의 최대 크기
EZFS_MAX_NAME_LEN	64	EZFS파일이름 길이, byte
TASK	2	태스크(Task)
TCP	5	TCP세션 개수
UDP	5	UDP세션 개수
TCP_RXBUF_SIZE	1,068	TCP 수신버퍼 크기, byte
TCP_TXBUF_SIZE	1,152	TCP 송신버퍼 크기, byte
PDB_TXBUF_SIZE	2,048	디버거 송신버퍼 크기, byte
HTTP_TXBUF_SIZE	1,536	HTTP 송신버퍼 크기, byte
UART_RXBUF_SIZE	1,024	UART 수신버퍼 크기, byte
UDP_RXBUF_SIZE	512	UDP 수신버퍼 크기, byte
ST	8	소프트웨어 타이머

표 10-2 펌웨어 사양

10.3 제한사항

구분	제한사항
네임 스페이스 레벨	PHP_MAX_NAME_SPACE - 1
함수 호출 레벨	PHP_MAX_NAME_SPACE - 2
사용자 식별자 길이	PHP_NAME_LEN - 1
문자열변수 최대 길이	PHP_MAX_STRING_LEN - 2
배열 오프셋 최대 크기	string length - 2
파일이름 길이	EZFS_MAX_NAME_LEN - 1
system함수 인수 길이	PHP_LLSTR_BLK_SIZE - 1
pid_ioctl함수 인수 길이	PHP_LLSTR_BLK_SIZE - 1
sendto함수 주소 길이	PHP_LLSTR_BLK_SIZE - 1
str_replace함수 $needle & replace 길이	PHP_LLSTR_BLK_SIZE - 1
inet_pton함수 주소 길이	PHP_LLSTR_BLK_SIZE - 1
inet_ntop함수 주소 길이	PHP_LLSTR_BLK_SIZE - 1
explode함수 구분자 길이	PHP_LLSTR_BLK_SIZE - 1
최대 UDP 수신가능 크기	UDP 수신버퍼 크기 - 2

표 10-3 제한사항

11 부록: pid_ioctl 명령어 인덱스

디바이스	명령어	인자 / 값	페이지
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 -

표 11-1 pid_ioctl 명령어 인덱스

12 문서 변경 이력

날짜	버전	변경내용	작성자
2014.09.23	1.0	○ 최초 작성	이 인
2015.08.07	1.1	○ 문서 제목 수정: for P20 추가 ○ TCP API 추가: TELNET, SSH, 웹소켓 ○ ST 출력모드 관련 내용 추가 ○ 부록 개선 ○ 일부 오류 수정 및 표현 개선	이 인
2015.11.03	1.2	○ 일부 오류 수정 및 표현 개선	이 인