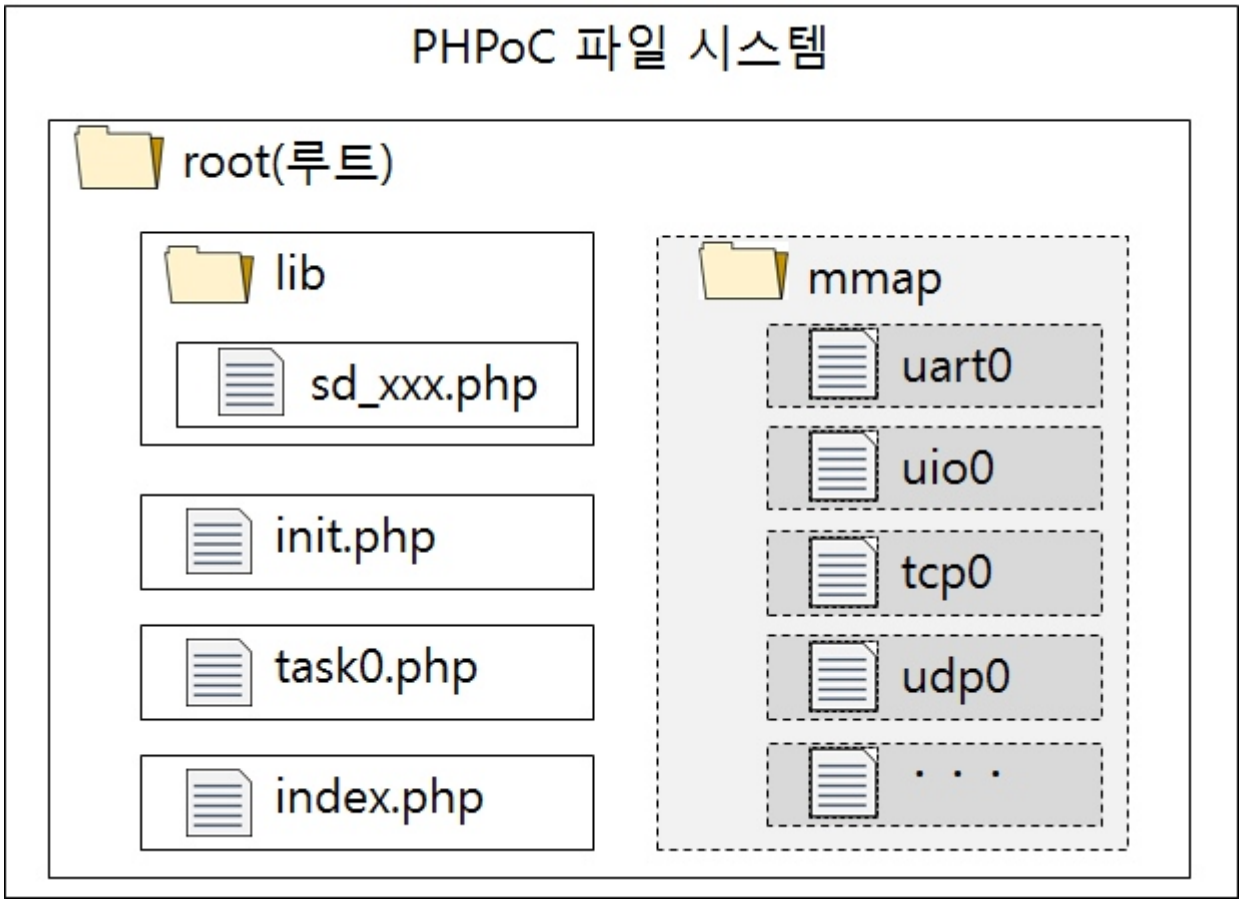
디바이스

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

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/DEVICE_NAME
```

※ PHPoC에서 사용자가 접근할 수 있는 디렉터리로는 root, /lib 그리고 /mmap이 있습니다. 사용자는 파일시스템에 임의로 디렉터리를 만들거나 삭제할 수 없습니다.

디바이스 종류

PHPoC는 다음과 같은 디바이스를 제공합니다.

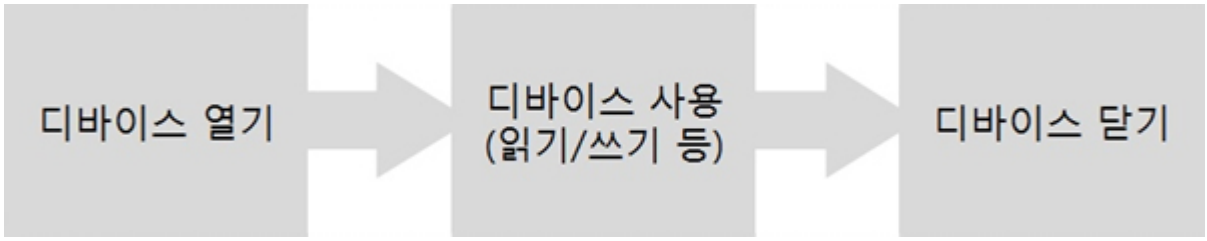
구분	디바이스 이름
하드웨어	디지털I/O(입/출력), UART(시리얼), NET(네트워크), ADC(아날로그 입력), I2C, SPI, HT(하드웨어타이머), RTC(내장 시계)
소프트웨어	TCP, UDP, ST(소프트웨어타이머), ENVS(시스템ENV), ENVU(사용자ENV), UM(사용자메모리), NM(비휘발성메모리)

※ 제공되는 디바이스들의 개수는 제품 종류 또는 펌웨어 버전에 따라서 다를 수 있습니다.

※ 제품 별 디바이스에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

디바이스 사용 절차

일반적인 디바이스 사용법은 다음과 같습니다.



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.

디바이스 사용

디바이스를 성공적으로 연 다음에는 반환 된 pid가 가리키는 디바이스를 사용할 준비가 된 것입니다. 디바이스 종류에 따라서 pid_ioctl, pid_read 등 관련된 함수들을 사용해 디바이스를 사용합니다.

디바이스 닫기

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

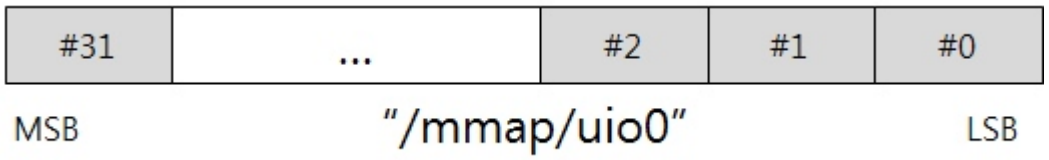
※ 주의: 디바이스를 닫은 이후에도 해당 디바이스에서 사용 된 물리적인 포트는 제품이 리부팅 되기 전까지 다른 디바이스에서 접근할 수 없습니다. 즉, 하나의 물리적인 포트는 제품이 리부팅하여 초기화되지 않는 한, 두 개 이상의 디바이스에서 사용될 수 없습니다.

디지털 I/O 개요

디지털 I/O는 디지털 입력을 감시하거나 출력을 제어하는 용도로 사용될 수 있습니다. 또한 제품의 각종 상태를 나타내기 위한 LED로 연결할 때에도 사용됩니다.

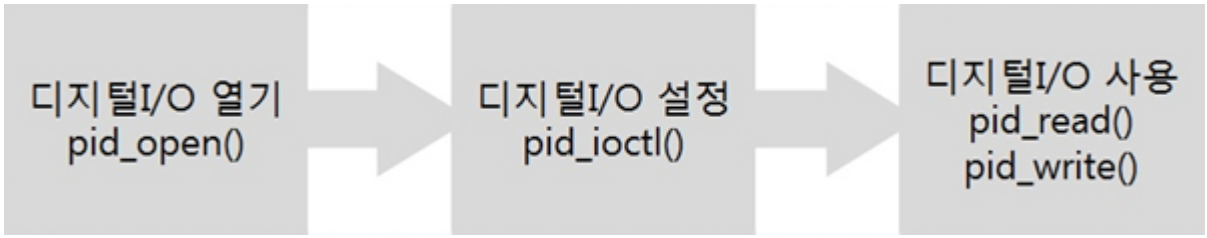
디지털 I/O의 구조

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



디지털 I/O 사용 절차

일반적인 디지털 I/O 사용 절차는 다음과 같습니다.



디지털 I/O 열기

디지털 I/O를 열기 위해서는 pid_open함수를 사용합니다.

```
<?php
$pid = pid_open("/mmap/uio0");      // 디지털 I/O 열기
?>
```

※ 제품 별 디지털 I/O에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

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.

입/출력 설정

TYPE에 설정 가능한 디지털 I/O의 입/출력 종류는 다음과 같습니다.

TYPE		설명
in		디지털 입력
in_pu		디지털 입력: 풀 업(Pull-Up)
in_pd		디지털 입력: 풀 다운(Pull-Down)
out	-	디지털 출력
	low	디지털 출력: 초기 값 LOW
	high	디지털 출력: 초기 값 HIGH
	toggle	디지털 출력: 초기 값 TOGGLE
out_pp	-	디지털 출력: 푸쉬 풀(Push-Pull)
	low	디지털 출력: 푸쉬 풀 + 초기 값 LOW
	high	디지털 출력: 푸쉬 풀 + 초기 값 HIGH
	toggle	디지털 출력: 푸쉬 풀 + 초기 값 TOGGLE
out_od	-	디지털 출력: 오픈 드레인(Open-Drain)
	low	디지털 출력: 오픈 드레인 + 초기 값 LOW
	high	디지털 출력: 오픈 드레인 + 초기 값 HIGH
	toggle	디지털 출력: 오픈 드레인 + 초기 값 TOGGLE

입력포트 풀 업(Pull-Up)

입력포트 풀 업은 입력포트의 유휴 상태를 HIGH로 만들 때 사용합니다. 입력포트를 풀 업 시키기 위해서는 디지털 입력의 TYPE을 in_pu로 설정합니다.

입력포트 풀 다운(Pull-Down)

입력포트 풀 다운은 입력포트의 유휴 상태를 LOW로 만들 때 사용합니다. 입력포트를 풀 다운 시키기 위해서는 디지털 입력의 TYPE을 in_pd로 설정합니다.

출력포트 푸쉬 풀(Push-Pull)

출력포트 푸쉬 풀은 출력포트의 상태가 ON일때 HIGH를, OFF일때 LOW를 출력하는 가장 기본적인 출력 모드입니다. 출력포트를 푸쉬 풀 방식으로 사용하기 위해서는 디지털 출력의 TYPE을 out_pp로 설정합니다.

출력포트 오픈 드레인(Open-Drain)

출력포트 외부에 전원공급원을 연결하고자 할 때 이 방식을 사용합니다. 만약 이 방식을 사용할 때 외부

에 전원공급원을 연결하지 않으면 PHPoC의 출력이 OFF일때는 포트 외부 출력이 LOW가 되지만 PHPoC의 출력이 ON일때는 외부 출력이 알 수 없는(Unknown) 상태가 됩니다. 따라서 외부 저항을 이용해 해당 포트를 풀 업(Pull-Up) 해줘야 합니다.

출력포트를 오픈 드레인 방식으로 사용하기 위해서는 디지털 출력의 TYPE을 out_od로 설정합니다.

LED 설정

디지털 I/O는 LED로 설정할 수 있습니다. TYPE에 설정 가능한 LED 종류는 다음과 같습니다.

TYPE	설명
led_sts	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

※ 각 led Type은 두 개 이상의 출력 핀에 동시에 설정할 수 없습니다.

디지털 I/O종류 설정 예

```
<?php
$pid = pid_open("/mmap/uio0");                  // 디지털 I/O 열기
pid_ioctl($pid, "set 0 mode in");              // 0번 입력 설정
pid_ioctl($pid, "set 1 mode in_pu");           // 1번 입력 설정: 풀 업
pid_ioctl($pid, "set 2 mode in_pd");           // 2번 입력 설정: 풀 다운
pid_ioctl($pid, "set 3-6 mode out");           // 3~6번 출력 설정
pid_ioctl($pid, "set 7-9 mode out high");      // 7~9번 출력 설정: HIGH
pid_ioctl($pid, "set 10 mode out low");        // 10번 출력 설정: LOW
pid_ioctl($pid, "set 11 mode out_pp high");    // 11번 출력 설정: 푸쉬 풀
pid_ioctl($pid, "set 12 mode out_od low");     // 12번 출력 설정: 오픈 드레인
pid_ioctl($pid, "set 13 mode led_net0_link"); // 13번 NET 링크 LED 설정
pid_ioctl($pid, "set 14 mode led_net0_rx");    // 14번 NET 수신 LED 설정
pid_ioctl($pid, "set 15 mode led_net0_tx");    // 15번 NET 송신 LED 설정
?>
```

디지털 I/O 출력 제한 설정

pid_ioctl함수를 이용해 디지털 I/O의 출력을 제한하거나 허용할 수 있습니다. 출력 제한이 설정되면 설정이 해제되기 전 까지 해당 포트로의 출력 명령은 적용되지 않습니다.

```
pid_ioctl($pid, "set N1[-N2] lock"); // 출력 제한
pid_ioctl($pid, "set N1[-N2] unlock"); // 출력 허용
```

※ 주의: 디지털 I/O는 기본적으로 출력 허용 상태입니다. 단, ST, UART, SPI 및 I2C등 디지털 I/O포트를 같이 사용하는 디바이스들이 사용 되었거나 이미 led 유형으로 설정 된 디지털 I/O포트는 출력 제한이 자동으로 설정 됩니다.

디지털 I/O 사용

디지털 I/O 상태 읽기

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

```
pid_read($pid, VALUE); // 모든 포트 상태 읽기(32 bits 단위)
pid_ioctl($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
		UART, SPI 및 I2C가 사용중인 핀: "hdev"
		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)	

여러 포트 상태 읽기 예

아래 예제는 UIO0의 여러 포트를 입력포트(풀업)로 설정한 후 그 상태 값을 읽어서 출력합니다.

```
<?php
$value = 0;
$pid = pid_open("/mmap/uio0");          // 디지털 I/O 열기
pid_ioctl($pid, "set 0-7 mode in_pu"); // 디지털 I/O 0~7번 입력(풀업) 설정
pid_read($pid, $value);                 // 디지털 I/O 상태 읽기(32bits 단위)
printf("0x%x₩r₩n", $value);             // 출력 결과 예: 0xffffffff
?>
```

단일 포트 정보 읽기 예

아래 예제는 UIO0의 0번 포트를 기본 상태가 HIGH인 출력포트로 설정하고 설정 유형 및 출력 상태를 읽어서 출력합니다.

```
<?php
$pid = pid_open("/mmap/uio0");              // 디지털 I/O 열기
pid_ioctl($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.

디지털 I/O에 값 쓰기

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

```
pid_write($pid, VALUE); // 여러 포트 출력(32 bits 단위)
pid_ioctl($pid, "set N output TYPE"); // 단일 포트 출력(1 bit 단위)
```

여러 포트 출력 예

아래 예제는 UIO0의 여러 포트를 출력포트로 설정한 후 임의의 값을 쓰고, 다시 디지털 I/O의 상태를 읽어 출력하는 예제 입니다.

```
<?php
$value = 0;
$pid = pid_open("/mmap/uio0");                          // 디지털 I/O 열기
pid_ioctl($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, ($value & 0xffffff00) | 0x00000055); // 0x00000055 출력
pid_read($pid, $value);                                 // 디지털 I/O 상태 읽기
printf("0x%08x₩r₩n", $value);                          // 출력 결과 예: 0x00000055
?>
```

단일 포트 출력 예

아래 예제는 UIO0의 0번을 기본 상태가 LOW인 출력포트로 설정하고 HIGH를 출력한 후 입/출력 상태를 읽어서 출력합니다.

```
<?php
$pid = pid_open("/mmap/uio0");                  // 디지털 I/O 열기
pid_ioctl($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

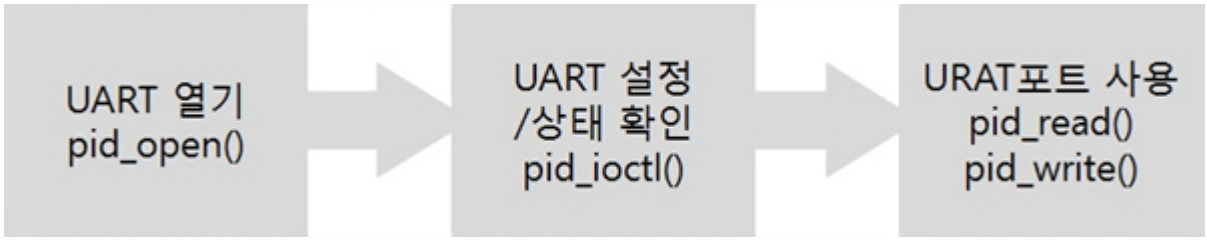
아래 예제는 0번 포트에 출력 제한을 설정한 경우와 그렇지 않은 경우에 HIGH 출력의 적용 여부를 비교하는 예제 입니다.

```
<?php
$pid = pid_open("/mmap/uio0");                  // 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
?>
```

UART 사용 절차

일반적인 UART 사용 절차는 다음과 같습니다.



UART 열기

UART를 열기 위해서는 pid_open함수를 사용합니다.

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

※ 제품 별 UART에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

UART 설정

UART는 사용하기 전에 반드시 설정이 필요합니다. 통신속도, 데이터비트, 정지 비트, 패리티, 흐름제어 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.

설정 가능한 UART 종류

ITEM	VALUE	설명	기본값
baud	예) 9600	통신속도[bps]	19200
parity	0	패리티 사용 안 함	0
	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	8
	7	데이터 비트 7(이 때 반드시 패리티를 사용해야 함)	
stop	1	정지 비트 1	1
	2	정지 비트 2	
flowctrl	0	흐름제어 사용 안 함	0
	1	RTS/CTS 사용	
	2	Xon/Xoff 사용	
	3	TxDE 사용(RS485)	

UART 설정 예

```
<?php
$pid = pid_open("/mmap/uart0");      // 0번 UART 열기
pid_ioctl($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
?>
```

UART 상태정보 확인

pid_ioctl함수의 get명령어로 UART의 각종 상태를 확인 할 수 있습니다.

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

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
count tx	누적 송신 데이터 카운트[Byte]	예) 65535	정수
rxbuf	수신버퍼 크기[Byte]	예) 1024	정수
rxlen	수신 데이터 크기[Byte]	예) 10	정수
count rx	누적 수신 데이터 카운트[Byte]	예) 65535	정수

UART 상태정보 확인 예

UART의 현재 설정 값은 다음과 같이 확인 할 수 있습니다.

```
<?php
$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
?>
```

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.

```
<?php
$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);
?>
```

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.

수신버퍼 여유공간

UART의 수신버퍼 여유공간은 다음과 같이 계산할 수 있습니다.

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

사용 예

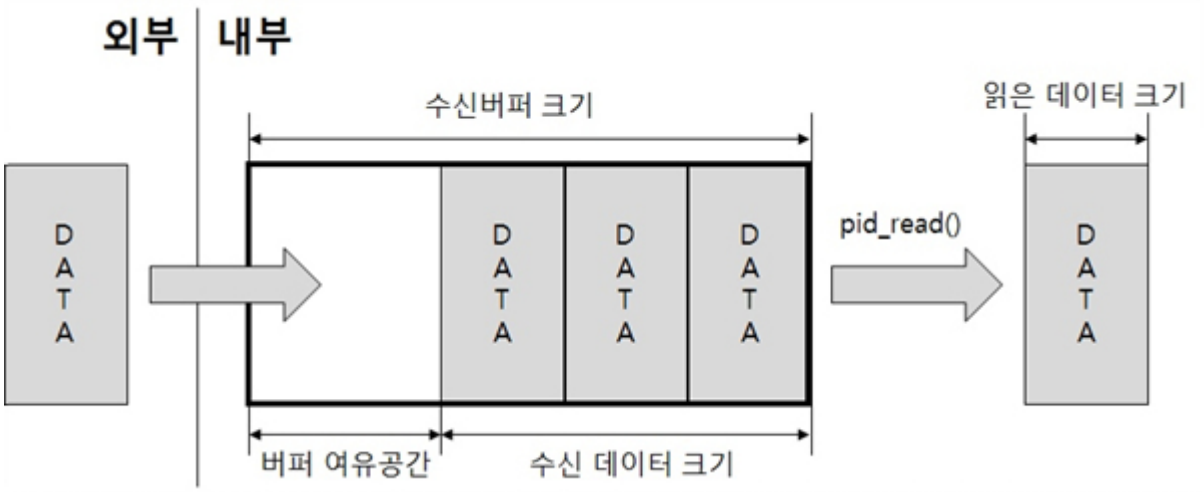
이 예제는 0번 UART의 수신버퍼의 여유공간을 계산하여 출력합니다.

```
<?php
$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);
?>
```

UART 사용

데이터 수신

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



pid_read함수는 다음과 같이 사용합니다.

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

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

사용 예

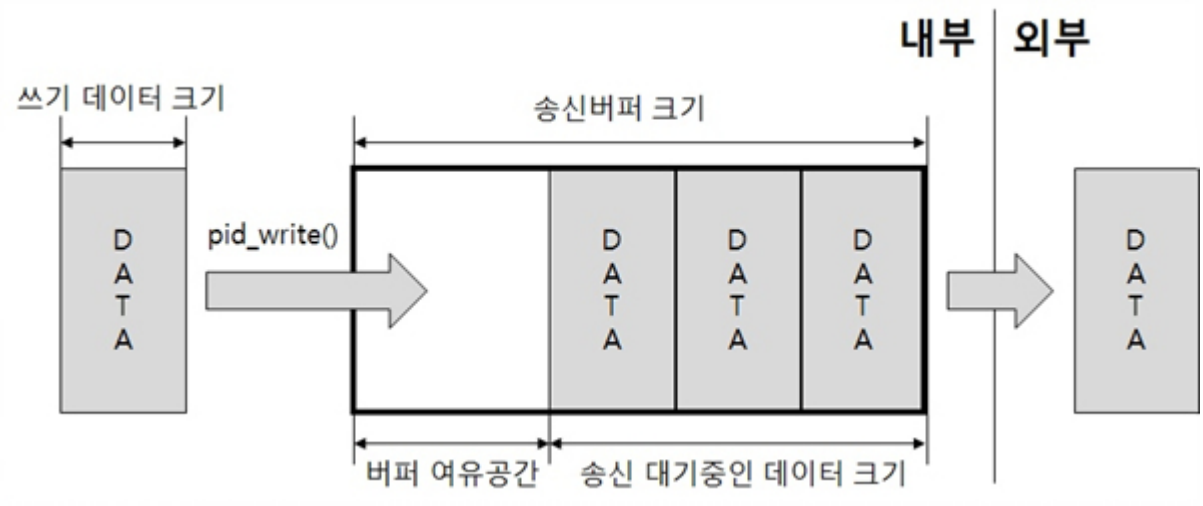
이 예제는 약 1초마다 UART로 수신되는 데이터를 확인하여 출력합니다.

```
<?php
$pid = pid_open("/mmap/uart0");                 // 0번 UART 열기
pid_ioctl($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)
{
    $rdata = "";
    $len_tot = pid_ioctl($pid, "get count rx"); // 누적 수신 데이터 카운트 확인
    $rxlen = pid_ioctl($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[total] = $len[$len_tot] / ";      // 읽은 데이터 크기 출력
    echo "rdata = $rdata₩r₩n";                  // 읽은 데이터 출력
```

```
    sleep(1);
}
pid_close($pid);
?>
```

데이터 송신

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



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.

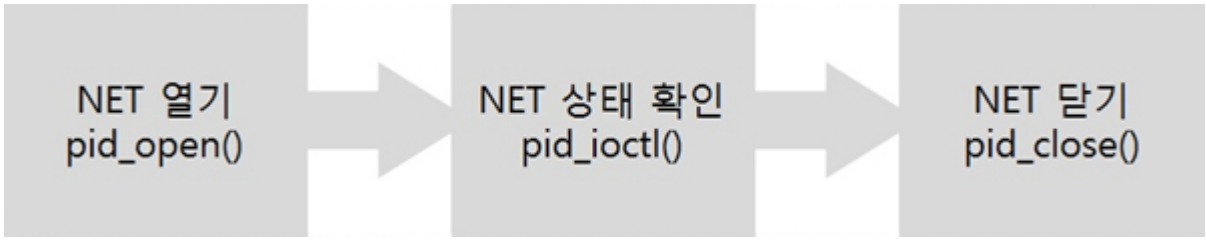
```
<?php
$len_tot = 0;
$sdata = "0123456789";
$pid = pid_open("/mmap/uart0");                 // 0번 UART 열기
pid_ioctl($pid, "set baud 9600");               // 통신속도 9600bps
$txbuf = pid_ioctl($pid, "get txbuf");          // 송신버퍼 크기 확인
while(1)
{
    $len_tot = pid_ioctl($pid, "get count tx"); // 누적 송신데이터 카운트 확인
    $txfree = pid_ioctl($pid, "get txfree");    // 송신버퍼 여유공간 확인
    echo "txfree = $txfree₩r₩n";                // 송신버퍼 여유공간 출력
    $len = pid_write($pid, $sdata, $txfree);    // 데이터 전송
    echo "len[total] = $len[$len_tot]₩r₩n";     // 전송한 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.

NET 사용 절차

일반적인 NET 사용 절차는 다음과 같습니다.



NET 열기

NET를 열기 위해서는 pid_open함수를 사용합니다.

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

※ 제품 별 NET에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

NET 설정

NET 설정은 pid_ioctl함수의 set명령을 통해 이루어집니다. NET 설정을 통해 Gratuitous ARP의 전송 간격과 횟수를 설정할 수 있습니다. PHPoC는 부팅 시 기본적으로 4개의 Gratuitous ARP를 2초 간격으로 전송합니다. 만약 이 동작을 수동으로 변경하고자 할 때 이 명령을 사용합니다.

```
pid_ioctl($pid, "set garp RC INTV");
```

RC는 전송 횟수를, INTV는 전송 간격(단위: 초)을 의미합니다.

Gratuitous ARP 설정 예

이 예제는 Gratuitous ARP 설정 예를 보여줍니다.

```
<?php
$pid = pid_open("/mmap/net0");       // NET 0 열기(이더넷)
pid_ioctl($pid, "set garp 5 10");    // GARP 설정: 10 초간격으로 5회 전송
?>
```

NET 상태정보 확인

pid_ioctl함수의 get명령어를 이용하여 NET의 상태정보를 확인할 수 있습니다.

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

ITEM은 확인 가능한 상태정보의 이름입니다.

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

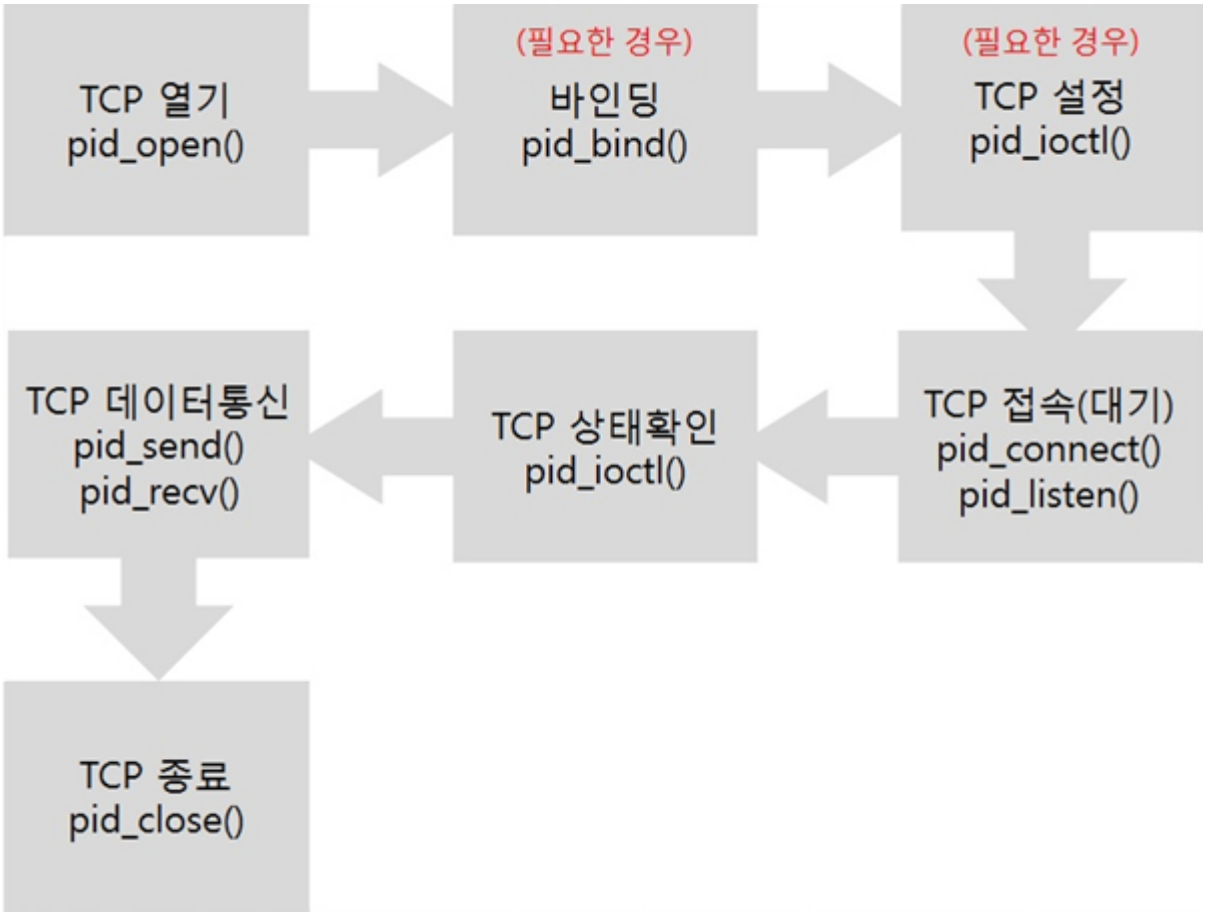
NET 상태정보 확인 예

이 예제는 NET의 각 상태정보를 확인하여 출력합니다.

```
<?php
$pid = pid_open("/mmap/net1");                  // 1번 NET 열기(무선랜)
echo pid_ioctl($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($pid, "get mode"), "₩r₩n";      // 무선랜 모드 확인
echo pid_ioctl($pid, "get speed"), "₩r₩n";     // 무선랜 속도 확인
pid_close($pid);                                // 네트워크포트 닫기
?>
```

TCP 사용 절차

일반적인 TCP 사용 절차는 다음과 같습니다.



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

TCP 열기

TCP를 열기 위해서는 pid_open함수를 사용합니다.

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

※ 제품 별 TCP에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

TCP 설정

TCP를 사용하기 전에 설정이 필요한 경우가 있습니다.
특히 SSL 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.

설정 가능한 TCP 항목

ITEM	VALUE		설명
nodelay	0		TCP Nagle 알고리즘 사용
	1		TCP Nagle 알고리즘 사용 안 함
api	ssl		SSL 사용
	tls		SSL 사용 ※ 펌웨어 1.5.0부터 사용 가능
	ws		웹 소켓(Web Socket) 서버 사용
ssl method	client		SSL클라이언트 (버전 자동) ※ 펌웨어 1.3.1부터 사용 가능
	server		SSL서버 (버전 자동) ※ 펌웨어 1.3.1부터 사용 가능
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

※ 주의 : "ssl method"는 "client" 또는 "server"를 사용하는 것을 권장합니다.(펌웨어 버전 1.3.1 이상 필요)

※ 주의 : "ssl method"에서 "ssl3_client"와 "ssl3_server"는 펌웨어 버전 1.5.0부터 더이상 지원하지 않습니다.

※ 주의 : "ssl method"에서 "tls1_client"와 "tls1_server"는 펌웨어 버전 2.1.0부터 더이상 지원하지 않습니다.

※ 주의 : "api"에서 "telnet"과 "ssh"는 펌웨어 버전 2.1.0부터 더이상 지원하지 않습니다.

※ 주의 : TCP Nagle 알고리즘은 데이터를 전송할 때 세그먼트의 수를 최소한으로 줄여 전송 효율을 높이기 위한 기능으로 약간의 지연시간이 동반됩니다.

※ 주의 : SSL은 TCP0 ~ 3에서만 사용이 가능합니다. 또한 해당 TCP아이디는 제품이 리부팅 되기 전까지 다른 api모드로 사용할 수 없습니다.

SSL사용

PHPoC는 다음 두 가지 명령을 통해 SSL 기능을 사용할 수 있습니다.

명령어	사용 가능한 펌웨어 버전
"set api ssl"	모든 버전
"set api tls"	1.5.0 이상

두 명령의 수행결과는 완전히 동일합니다.

다음은 SSL 서버 설정 예 입니다.

SSL 서버 설정 예

```
<?php
$port = 1470;                                   // 포트번호
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_ioctl($pid, "set api ssl");                 // SSL 사용
pid_ioctl($pid, "set ssl method server");       // SSL 서버 설정
pid_bind($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

```
<?php
$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($pid, "set ssl method client");       // SSL 클라이언트 설정
pid_bind($pid, "", 0);                          // 바인딩
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.

웹 소켓 사용

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

웹 소켓 서버 사용 예

이 예제는 웹 소켓이 접속되면 "hello, world"라는 문구를 매 초 전송합니다.

```
<?php
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_ioctl($pid, "set api ws");                  // 웹 소켓 설정
pid_ioctl($pid, "set ws path WebConsole");     // URI 경로: /WebConsole
pid_ioctl($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 text.phpoc");    // 프로토콜: text.phpoc
pid_bind($pid, "", 0);                          // 바인딩: 기본포트(80)

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

  pid_send($pid, "hello, world!₩r₩n");         // 데이터 송신
  sleep(1);
}
pid_close($pid);
?>
```

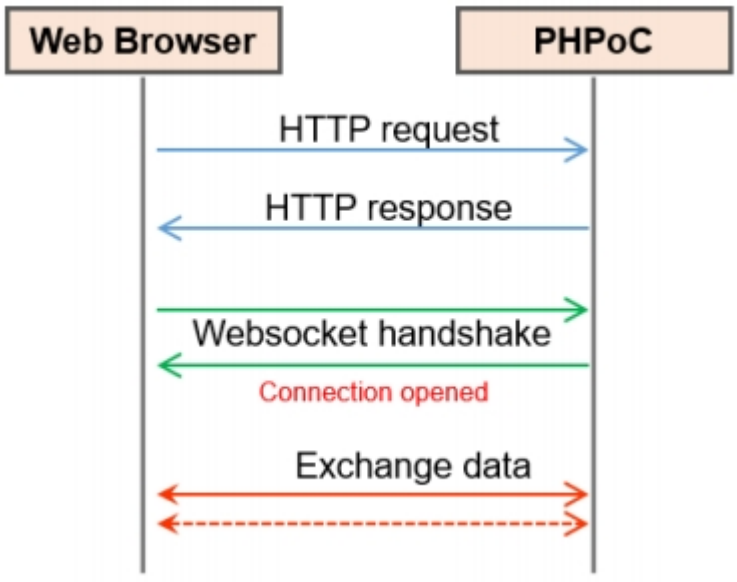
아래는 위 예제를 실행하기 위해서 필요한 웹페이지(index.php) 소스코드 입니다.

```
<html>
<head>
<title>PHPoC / <?echo system("uname -i")?></title>
<meta name="viewport" content="width=device-width, initial-scale=0.7">
<style>
body { text-align:center; }
textarea { width:400px; height:400px; padding:10px; font-family:courier; font-size:14px; }
 </style>
<script>
var ws;
var wc_max_len = 32768;
function ws_onopen()
{
    document.getElementById("ws_state").innerHTML = "OPEN";
    document.getElementById("wc_conn").innerHTML = "Disconnect";
}
function ws_onclose()
{
    document.getElementById("ws_state").innerHTML = "CLOSED";
```

```
    document.getElementById("wc_conn").innerHTML = "Connect";

    ws.onopen = null;
    ws.onclose = null;
    ws.onmessage = null;
    ws = null;
}
function wc_onclick()
{
    if(ws == null)
    {
        ws = new WebSocket("ws://<?echo _SERVER("HTTP_HOST")?>/WebConsole", "text.phpoc");
        document.getElementById("ws_state").innerHTML = "CONNECTING";

        ws.onopen = ws_onopen;
        ws.onclose = ws_onclose;
        ws.onmessage = ws_onmessage;
    }
    else
        ws.close();
}
function ws_onmessage(e_msg)
{
    e_msg = e_msg || window.event; // MessageEvent

    var wc_text = document.getElementById("wc_text");
    var len = wc_text.value.length;

    if(len > (wc_max_len + wc_max_len / 10))
        wc_text.innerHTML = wc_text.value.substring(wc_max_len / 10);

    wc_text.scrollTop = wc_text.scrollHeight;
    wc_text.innerHTML += e_msg.data;
}
function wc_clear()
{
    document.getElementById("wc_text").innerHTML = "";
}
</script>
</head>
<body>

<h2>
<p>
Web Console : <span id="ws_state">CLOSED</span><br>
</p>
<textarea id="wc_text" readonly="readonly"></textarea><br>
<button id="wc_conn" type="button" onclick="wc_onclick();">Connect</button>
<button id="wc_clear" type="button" onclick="wc_clear();">Clear</button>
</h2>

</body>
</html>
```

위 예제에서 웹 소켓 서버(php 스크립트)와 클라이언트(javascript)는 PHPoC에 구현되었지만 웹 소켓 서버는 PHPoC에서 실행되고 웹 소켓 클아이언트는 웹 브라우저에서 실행됩니다. 위 예제의 동작 흐름도

는 다음과 같습니다.



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

※ 웹 소켓을 PC에서 사용하기 위해서는 반드시 웹 소켓을 지원하는 웹 브라우저를 사용하시기 바랍니다.

TCP 접속

TCP클라이언트(능동 접속)

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

```
pid_bind($pid, "", 0);
pid_connect($pid, $addr, $port);
```

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

TCP클라이언트 사용 예

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

TCP서버(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

```
<?php
$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)
```

```
?>
```

TCP 상태정보 확인

pid_ioctl함수의 get명령어로 TCP의 각종 상태를 확인 할 수 있습니다.

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

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
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

TCP세션 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. SSL도 마찬가지로 접속 안 됨, 접속 완료 및 접속 대기의 세 가지 상태가 있습니다. 이 값들은 모두 PHPoC에서 미리 정의된 상수(predefined constant)입니다.
모든 세션의 상태는 다음과 같이 확인할 수 있습니다.

```
<?php
$state = pid_ioctl($pid, "get state");
?>
```

※ TCP 또는 SSL접속을 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.

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.

```
<?php
$tx_len = -1;
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_bind($pid, "", 0);                          // 바인딩
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
?>
```

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.

수신버퍼 여유공간

TCP 수신버퍼 여유공간은 다음과 같이 계산할 수 있습니다.

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

사용 예

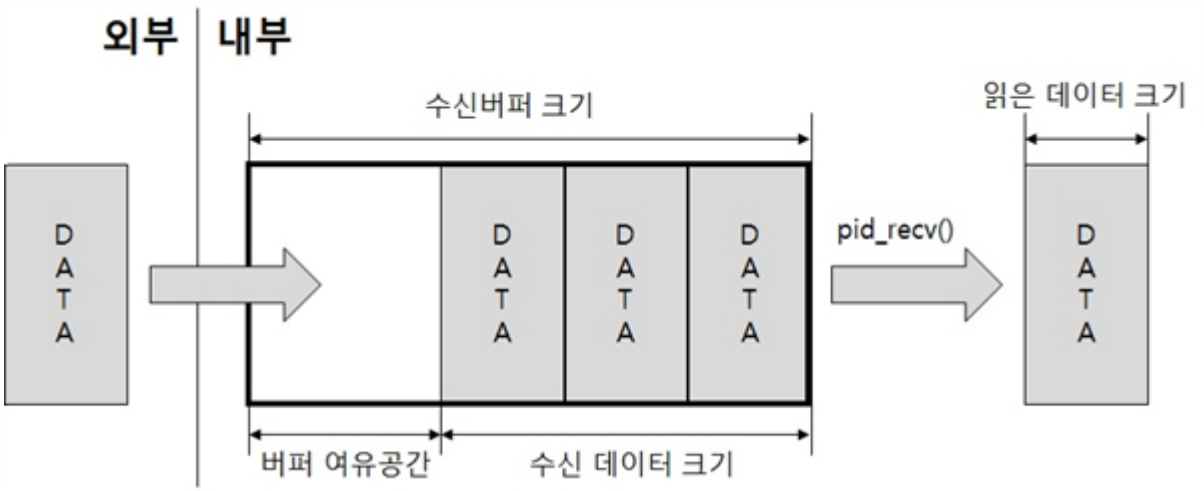
이 예제는 TCP접속 후 수신버퍼의 여유공간을 계산하여 출력합니다.

```
<?php
$rx_free = 1068;
$pid = pid_open("/mmap/tcp0");                // 0번 TCP 열기
pid_bind($pid, "", 0);                        // 바인딩
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
?>
```

TCP 데이터 통신

TCP 데이터 수신

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



pid_recv함수는 다음과 같이 사용합니다.

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

사용 예

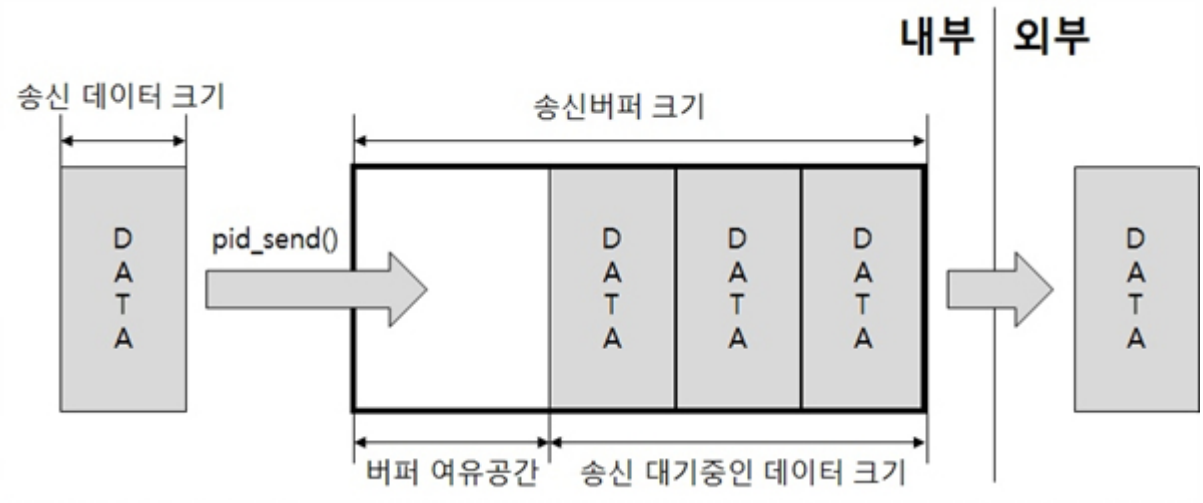
이 예제는 매 초마다 TCP로 수신되는 데이터를 확인하고 출력합니다.

```
<?php
$rdata = "";
$pid = pid_open("/mmap/tcp0");              // 0번 TCP 열기
pid_bind($pid, "", 0);                      // 바인딩
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);
```

```
?>
```

TCP 데이터 송신

pid_send함수를 이용해 송신한 데이터는 송신버퍼에 저장되었다가 네트워크로 전송됩니다.



pid_send함수는 다음과 같이 사용합니다.

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

사용 예

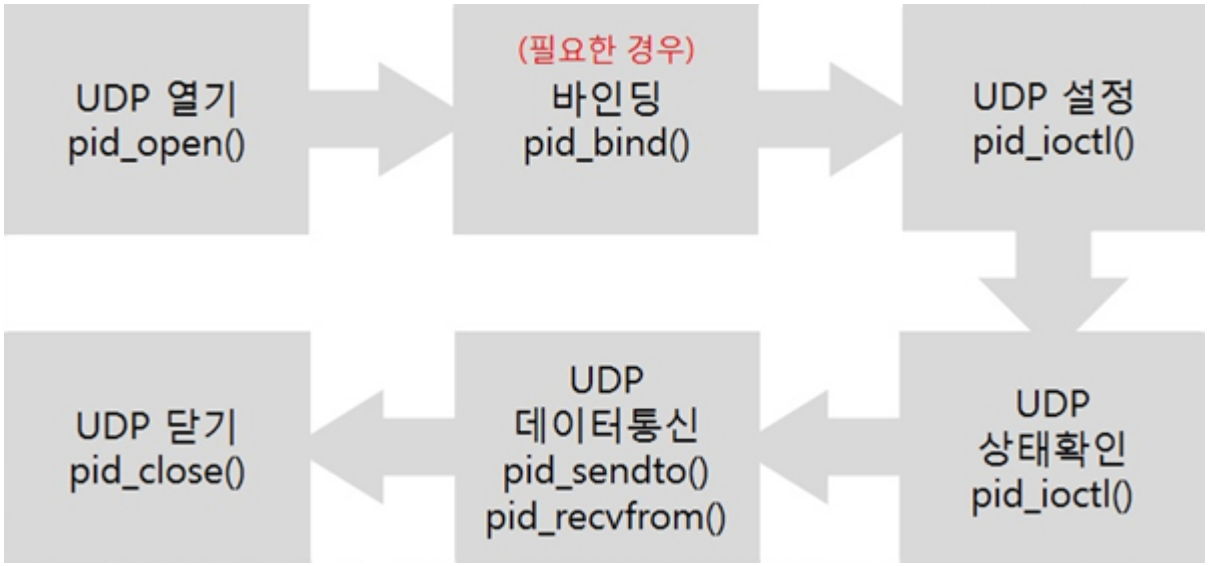
이 예제는 송신버퍼의 여유공간을 확인하여 TCP로 데이터를 송신합니다.

```
<?php
$sdata = "0123456789";
$pid = pid_open("/mmap/tcp0");                  // 0번 TCP 열기
pid_bind($pid, "", 0);                          // 바인딩
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.

UDP 사용 절차

일반적인 UDP 사용 절차는 다음과 같습니다.



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

UDP 열기

UDP를 열기 위해서는 pid_open함수를 사용합니다.

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

※ 제품 별 UDP에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

UDP 바인딩

UDP데이터를 수신하기 위해서는 pid_bind함수를 사용하는 바인딩 과정이 필요합니다.

```
<?php
$pid = pid_bind($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

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

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.

설정 가능한 UDP 항목

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

UDP 설정 예

```
<?php
$pid = pid_open("/mmap/udp0");              // 0번 UDP 열기
pid_bind($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
?>
```

UDP 상태정보 확인

pid_ioctl함수의 get명령어로 UDP의 각종 상태를 확인 할 수 있습니다.

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

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

수신 데이터 크기 확인

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

```
<?php
$rxlen = pid_ioctl($pid, "get rxlen");
?>
```

사용 예

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

```
<?php
$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);
?>
```

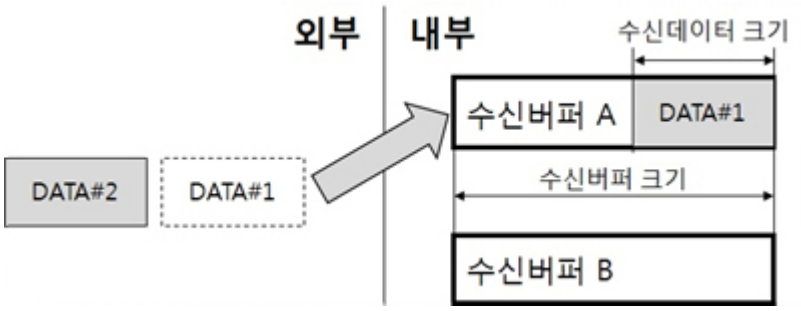
UDP 데이터 통신

데이터 수신

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

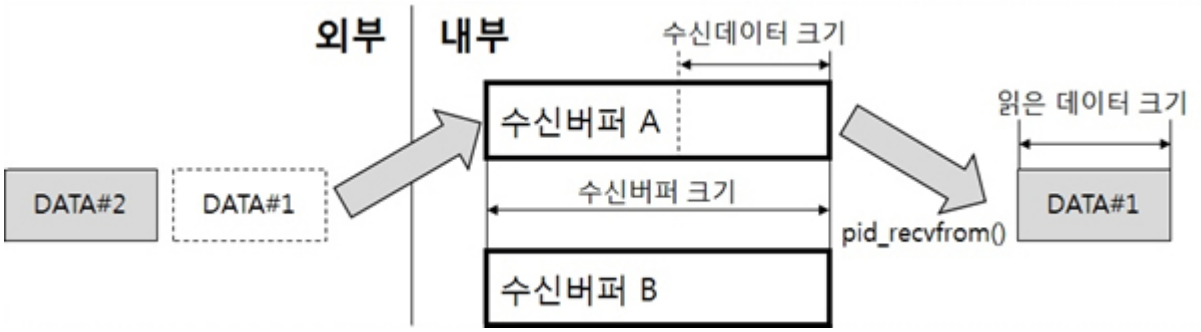
※ 제품 별 UDP 수신버퍼의 크기는 부록을 참조하시기 바랍니다.

네트워크로부터 데이터 수신



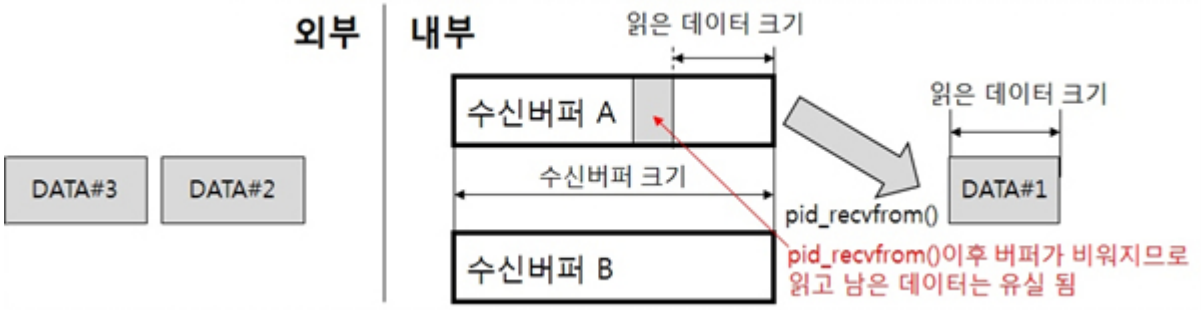
수신버퍼에 저장 된 데이터 읽기

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



수신버퍼에 저장 된 데이터의 크기보다 작은 길이만큼 읽은 경우

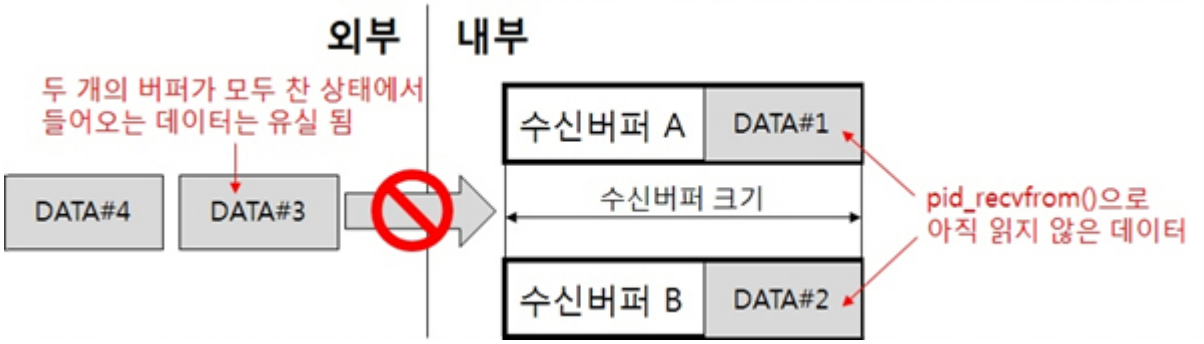
수신버퍼에 읽고 남은 데이터는 버퍼를 비움과 동시에 유실됩니다.



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.



사용 예

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

```
<?php
$rbuf = "";
$pid = pid_open("/mmap/udp0");              // 0번 UDP 열기
pid_bind($pid, "", 1470);                   // 바인딩
while(1)                                    // 무한 루프
{
    $rxlen = pid_ioctl($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);
}
?>
```

데이터 송신

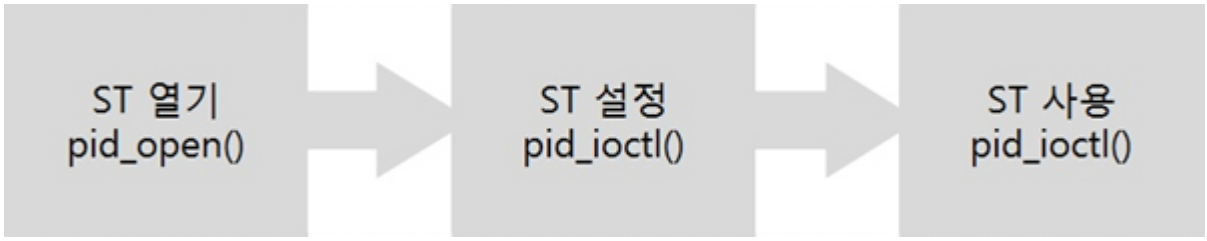
UDP 데이터를 송신하기 위해서는 pid_sendto함수를 사용합니다.

UDP 데이터 송신 예

```
<?php
$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);
?>
```

ST 사용 절차

일반적인 ST 사용 절차는 다음과 같습니다.



ST 열기

ST를 열기 위해서는 pid_open함수를 사용합니다.

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

※ 제품별 제공되는 ST에 대한 정보는 부록을 참조하시기 바랍니다.

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

공통 명령어

다음 명령어들은 ST의 동작모드와 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

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");

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"");

초기화

"reset"명령어는 다음과 같은 동작을 합니다:

- ST의 동작을 그 즉시 중단하고 초기화 합니다.
- ST 출력핀에 LOW를 출력합니다.

구분	문법
초기화	pid_ioctl($pid, "reset");

상태 읽기

"get state" 명령어는 ST의 상태를 읽는 명령어 입니다.

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

이 명령어에 의한 반환 값은 다음과 같습니다.

반환 값	상태
0	정지
1 ~ 5	동작 중

시작

ST를 시작시키기 위해서는 "start"명령을 사용합니다.

구분	문법
시작	pid_ioctl($pid, "start");

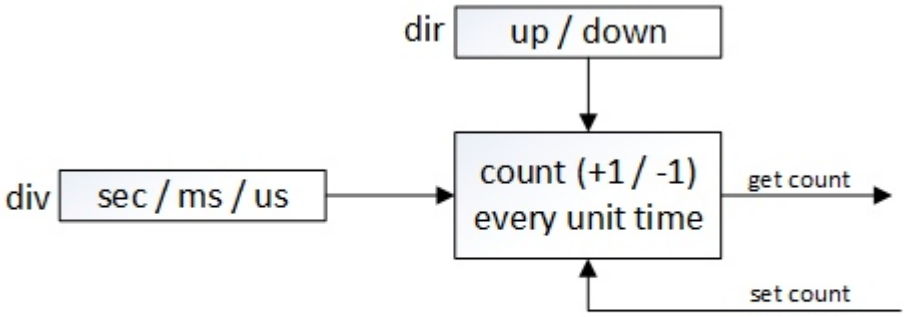
정지

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

구분	문법
정지	pid_ioctl($pid, "stop");

프리모드

프리모드는 ST를 일반적인 카운터로 동작시키는 모드 입니다.



ST(free) Block Diagram

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

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");

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");

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
마이크로 초 단위	0 ~ (2^{63} - 1)
밀리 초 단위	0 ~ (2^{63} - 1) / 1,000
초 단위	0 ~ (2^{63} - 1) / 1,000,000

카운트 값 읽기

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

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

프리모드 사용 예

프리모드에서의 ST값은 pid_ioctl함수의 "get count"명령으로 읽을 수 있습니다.

```
<?php
$tick = pid_ioctl($pid, "get count");
?>
```

업카운터

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

```
<?php
$pid = pid_open("/mmap/st0");               // 0번 ST 열기
pid_ioctl($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.

```
<?php
$pid = pid_open("/mmap/st0");               // 0번 ST열기
pid_ioctl($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);
?>
```

토글출력모드

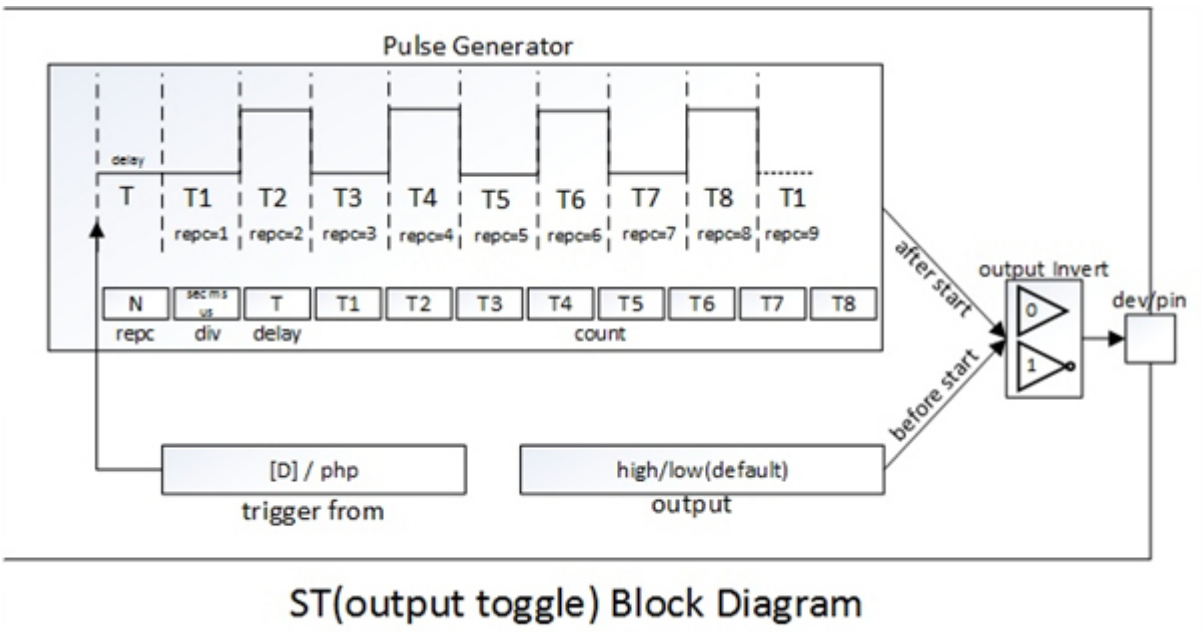
개요

이 모드는 다양한 지속시간을 갖는 하나 또는 여러개의 직사각형 펄스를 출력하는데 사용합니다. 이 모드에서 ST핀은 미리 정의된 시간 간격에 따라 출력이 토글됩니다.

ST의 출력파형은 다음에 따라 달라집니다:

- 타이머가 시작되는 시점의 ST출력핀의 상태
- 두 개의 연속적인 토글출력 사이의 지속시간(카운트 값)
- 반복 횟수

다음은 ST 토글출력모드의 블록다이어그램 입니다.



ST(output toggle) Block Diagram

명령어

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	od			오픈 드레인(Open-Drain)
		pp			푸쉬 풀(Push-Pull)
		low			LOW 출력
		high			HIGH 출력
		dev	uio0	#pin	출력 핀 설정: uio0의 #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

지연시간 설정

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

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

반복횟수 설정

토글출력모드에서 이 명령어는 토글 신호의 반복횟수를 설정합니다.

명령어	문법	N의 범위
set repc	pid_ioctl($pid, "set repc N");	0 ~ 10억(1,000,000,000)

반복횟수의 기본값은 0입니다. 이 값이 0이면 반복횟수는 최대 반복횟수(10억)로 설정됩니다.
(카운트 값 설정의 예제 파형 참조)

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.

명령어	문법
set count	pid_ioctl($pid, "set count T1 T2 ... T8");

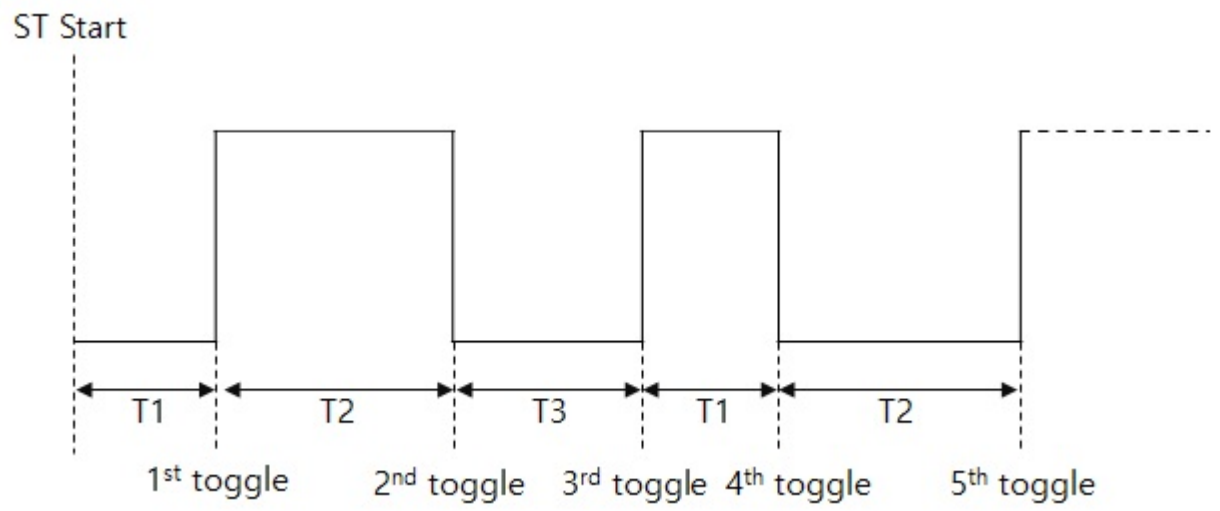
토글출력모드에서 가능한 카운트 값의 범위는 다음과 같습니다.

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

이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다. 토글출력모드에서

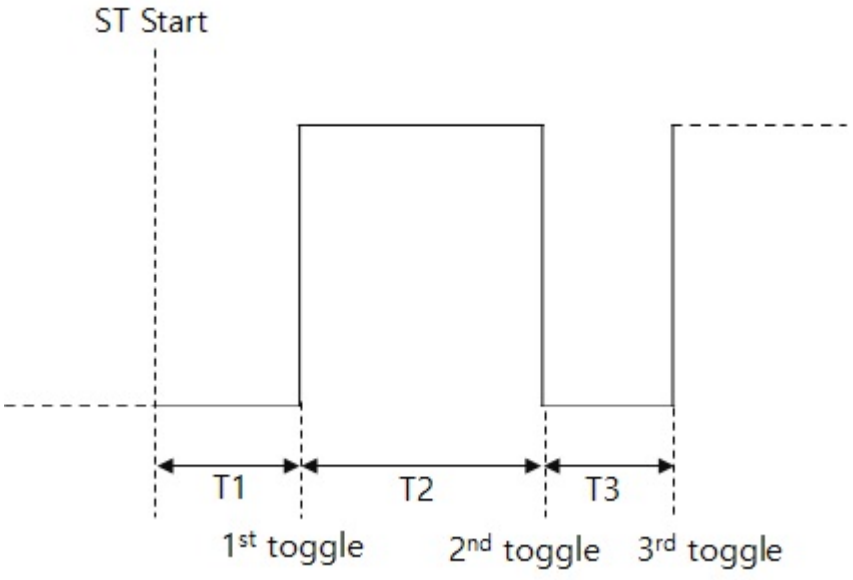
카운트 값을 2개 이상 설정하면 이 값들은 출력 시점마다 순서대로 사용됩니다. 만약 반복횟수가 설정한 카운트 값의 수 보다 많으면 다시 첫 번째 값부터 순서대로 사용됩니다. 아래 그림은 다음 조건하에서의 출력 파형을 보여줍니다.

- "set count T1 T2 T3": 3개(T1, T2, T3)의 카운트 값 설정
- "set repc 5": 5번의 반복횟수 설정
- 타이머 시작 시점의 ST출력핀의 상태는 LOW

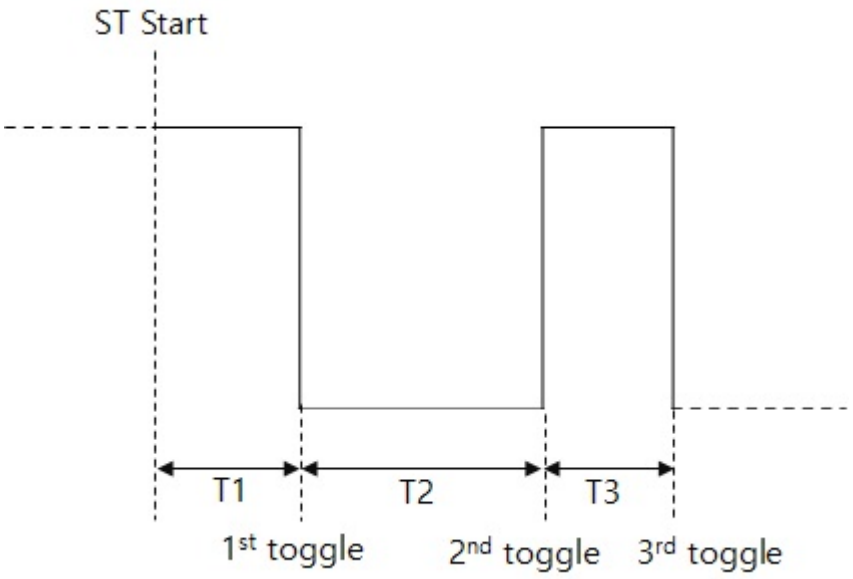


출력 파형은 타이머가 시작되는 시점의 ST 출력핀의 상태에 따라 달라집니다. 다음 예는 ST 출력핀의 상태가 다를 때의 파형을 보여줍니다. 반복 횟수는 3회이고 카운트 값은 각각 T1, T2 그리고 T3입니다.

- 타이머가 시작되는 시점에 ST 출력핀의 상태가 LOW일 때



- 타이머가 시작되는 시점에 ST 출력핀의 상태가 HIGH일 때



ST 출력핀의 상태는:

- "set output dev" 명령 이후에 LOW
- "reset" 명령 이후에 LOW
- "set output low"명령 직후에 LOW
- "set output high"명령 직후에 HIGH
- 타이머 출력 모드(토글, 펄스 또는 PWM)에 따라 LOW 또는 HIGH

두개의 연속적인 토글 출력 사이의 지속시간(카운트 값)은 "set count"명령으로 설정합니다.

출력핀 설정

명령어	문법
set output dev D N	pid_ioctl($pid, "set output dev uio0 0");

ST의 출력모드를 사용하기 전에 이 명령을 이용하여 출력핀을 지정해야 합니다. 디바이스 이름(예: uio0)과 핀 번호를 D와 N에 각각 지정합니다.

출력상태 설정 [low/high]

이 명령은 ST 출력핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	ST 출력핀에 LOW출력
set output high	pid_ioctl($pid, "set output high");	ST 출력핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 ST 출력핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

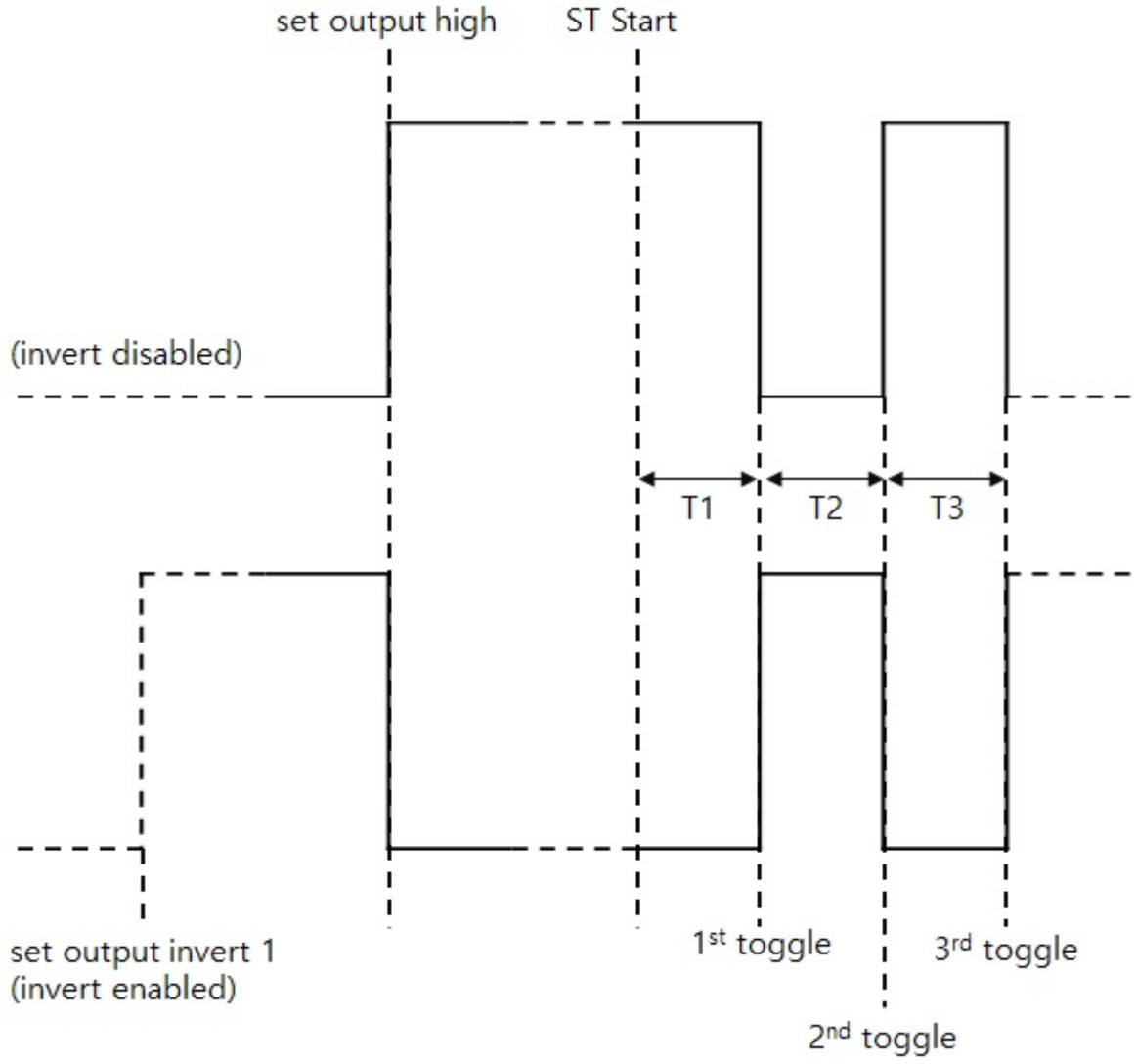
반전출력이 활성화되어 있으면:

- 출력핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.

반전출력 활성상태가 바뀌면(반전출력 활성화 또는 비활성화) ST 출력핀의 상태는 즉시 토글됩니다.

반전출력의 기본상태는 비활성화상태 입니다.

다음은 반전출력모드의 활성화/비활성화에 따른 파형의 차이를 보여줍니다. 반복 횟수는 3회이고 카운트 값은 각각 T1, T2 그리고 T3입니다. 타이머 시작 전 "set output high"명령이 사용되었습니다.



위 그림에서 보는것과 같이 반전출력이 활성화되면 "set output high"명령에 의한 출력이 LOW가 됩니다.

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");

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");

토글출력모드 사용 예

토글출력모드는 ST의 출력신호를 반전시키는 모드 입니다.

토글출력모드 사용

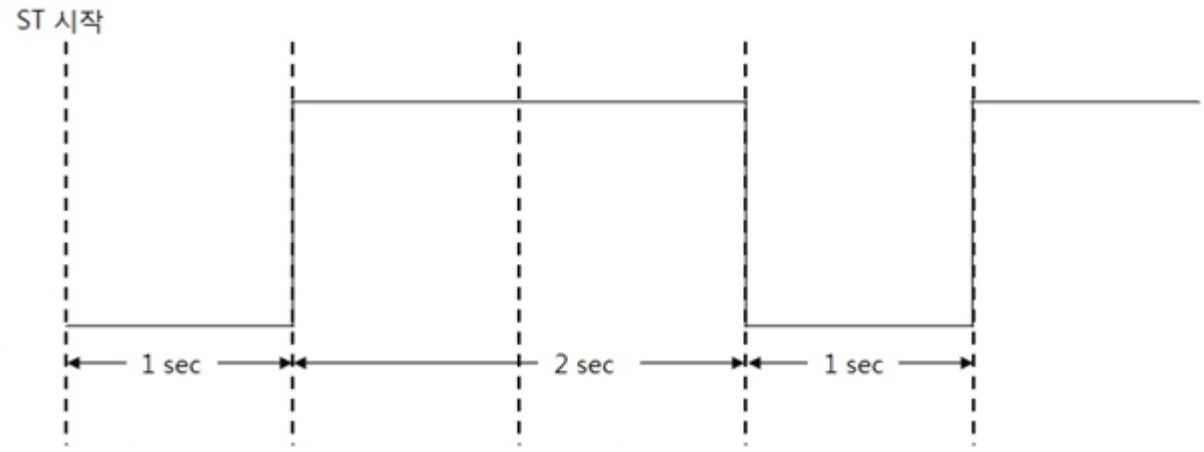
```
<?php
$pid = pid_open("/mmap/st0");                // 0번 ST 열기
pid_ioctl($pid, "set div sec");              // 단위 설정: 초
pid_ioctl($pid, "set mode output toggle");  // 토글출력모드 설정
pid_ioctl($pid, "set output dev uio0 0");   // 출력 핀 설정: uio0의 0번
pid_ioctl($pid, "set repc 1");               // 출력 횟수 설정: 1회
pid_ioctl($pid, "set count 1");              // T1 설정: 1초
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.

반복적 토글출력모드

```
<?php
$pid = pid_open("/mmap/st0");                // 0번 ST 열기
pid_ioctl($pid, "set div sec");              // 단위 설정: 초
pid_ioctl($pid, "set mode output toggle");  // 토글출력모드 설정
pid_ioctl($pid, "set output dev uio0 0");   // 출력 핀 설정: uio0의 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.



펄스출력모드

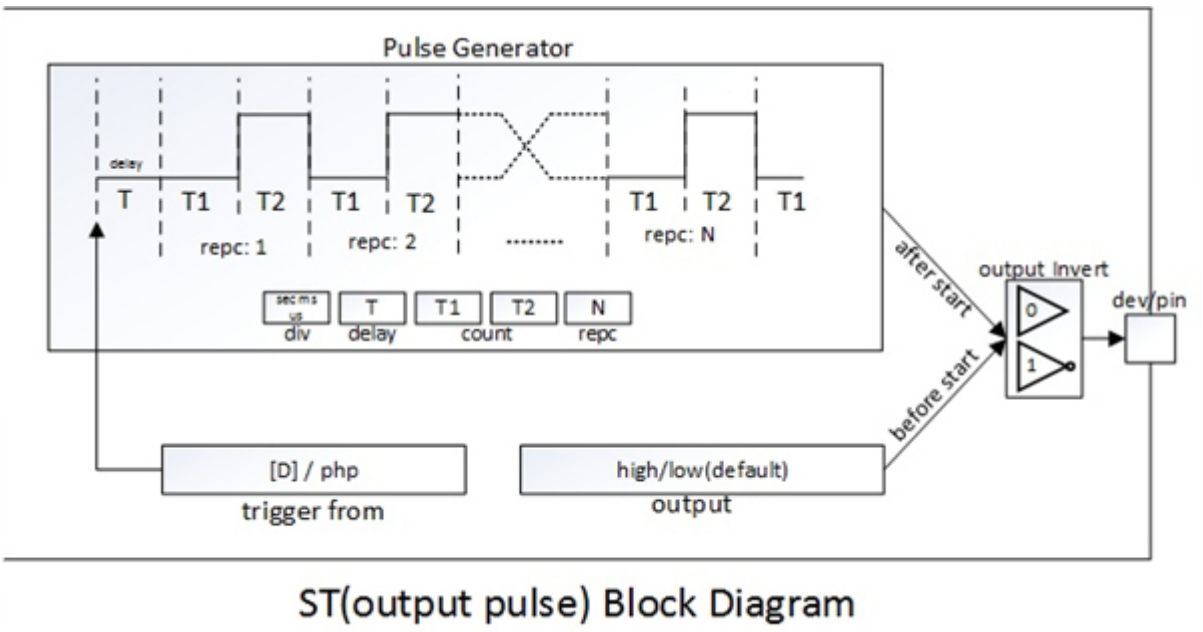
개요

이 모드는 하나 또는 여러개의 직사각형 펄스를 출력하는데 사용합니다. 이 모드에서 각 직사각형 펄스의 모양은 모두 동일합니다.

ST의 출력파형은 다음에 따라 달라집니다:

- 타이머가 시작되는 시점의 ST출력핀의 상태와는 무관
- low-level과 high-level의 지속시간(카운트 값)
- 반복 횟수

다음은 ST 펄스출력모드의 블록다이어그램 입니다.



ST(output pulse) Block Diagram

명령어

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	od			오픈 드레인(Open-Drain)
		pp			푸쉬 풀(Push-Pull)
		low			LOW 출력
		high			HIGH 출력
		dev	uio0	#pin	출력 핀 설정: uio0의 #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

지연시간 설정

펄스출력모드에서는 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");

반복횟수 설정

펄스출력모드에서는 출력 신호의 반복횟수를 설정할 수 있습니다.

명령어	문법	N의 범위
set repc	pid_ioctl($pid, "set repc N");	0 ~ 10억(1,000,000,000)

반복횟수의 기본값은 0입니다. 이 값이 0이면 반복횟수는 최대 반복횟수(10억)로 설정됩니다.
(카운트 값 설정의 예제 파형 참조)

카운트 값 설정

하나의 직사각형 펄스는 high-level 신호와 low-level 신호로 구성됩니다. 펄스출력모드에서는 이 명령어에 의해 직사각형 펄스의 low-level 및 high-level 유지시간을 설정할 수 있습니다. 카운트 값의 설정 단위는 "set div"명령에 의한 단위가 사용됩니다.

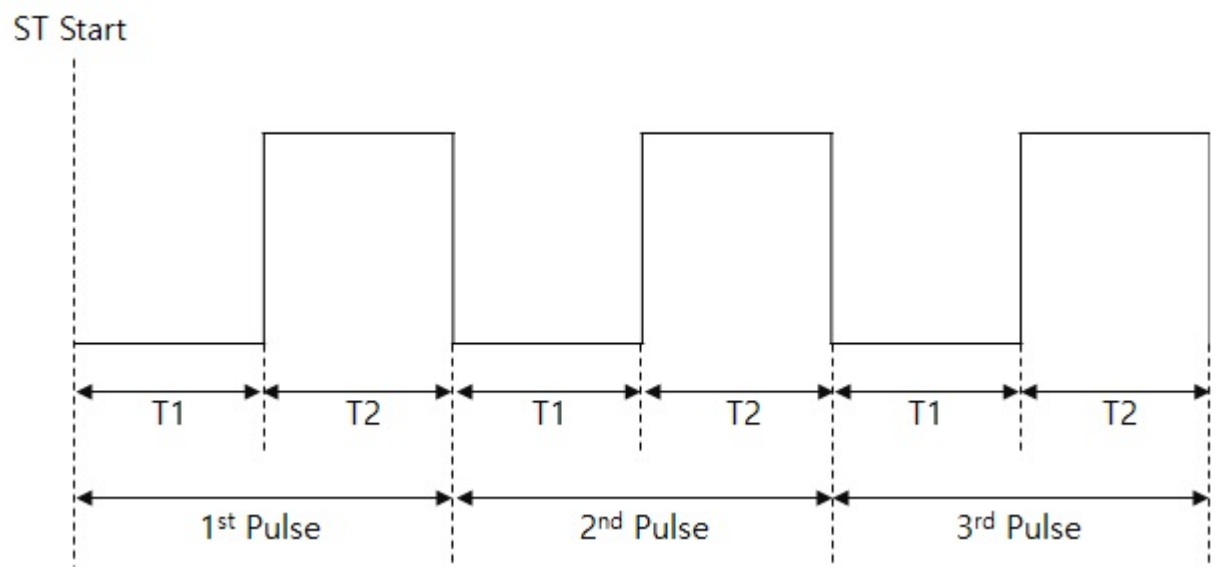
명령어	문법
set count	pid_ioctl($pid, "set count T1 T2");

펄스출력모드에서 설정 가능한 카운트 값 T1과 T2의 범위는 다음과 같습니다.

단위	T1, T2 설정 가능 범위(10μs ~ 30분)
마이크로 초	10 ~ 1,800,000,000
밀리 초	1 ~ 1,800,000
초	1 ~ 1,800

이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다. 아래 그림은 다음 조건하에서의 출력 파형을 보여줍니다.

- "set repc 3": 3번의 반복횟수 설정
- "set count T1 T2": 카운트 값 T1 및 T2 설정



출력핀 설정

명령어	문법
set output dev D N	pid_ioctl($pid, "set output dev uio0 0");

ST의 출력모드를 사용하기 전에 이 명령을 이용하여 출력핀을 지정해야 합니다. 디바이스 이름(예: uio0)과 핀 번호를 D와 N에 각각 지정합니다.

출력상태 설정 [low/high]

이 명령은 ST 출력핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	ST 출력핀에 LOW출력
set output high	pid_ioctl($pid, "set output high");	ST 출력핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 ST 출력핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

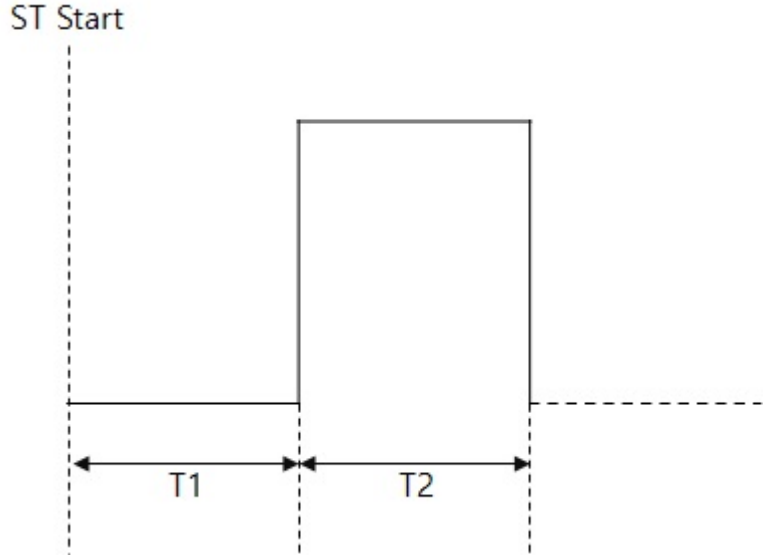
명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

반전출력이 활성화되어 있으면:

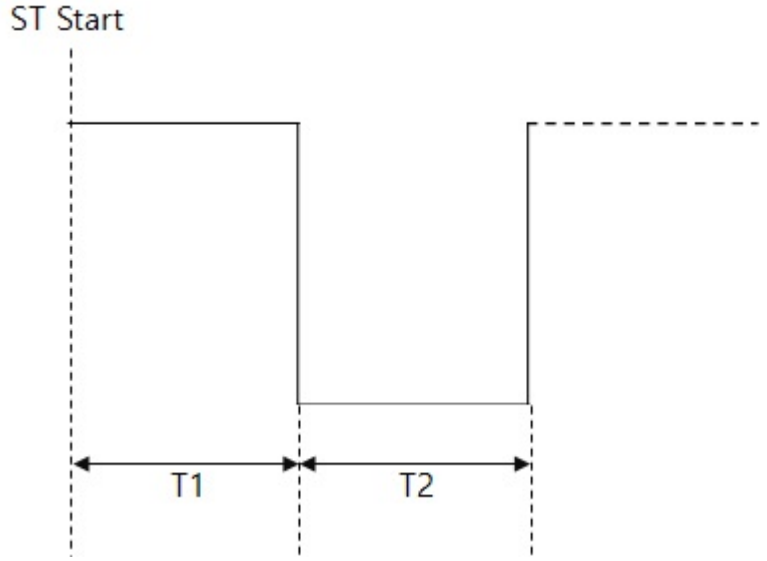
- 출력핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.

반전출력 활성상태가 바뀌면(반전출력 활성화 또는 비활성화) ST 출력핀의 상태는 즉시 토글됩니다. 즉, 이 명령어에 의해 펄스의 형태가 결정됩니다. 펄스는 다음 두가지의 형태가 있습니다.

- 반전 출력 비활성화



- 반전 출력 활성화



반전출력의 기본상태는 비활성화상태 입니다.

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 방법

은 다음과 같습니다.

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

ST의 트리거 설정 기본 값은 "대상 없음(php)" 입니다.

반복횟수 읽기

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

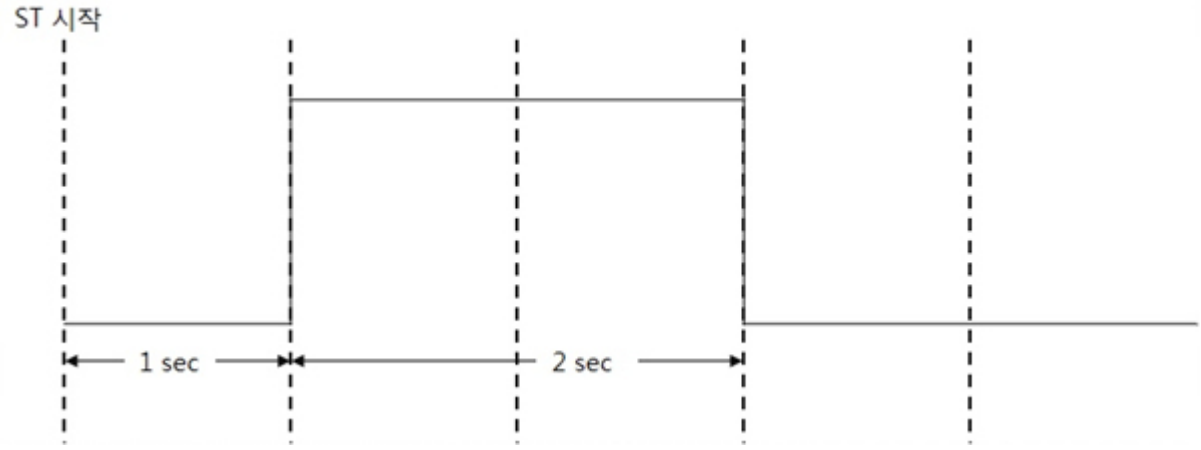
구분	문법
남은 출력 횟수	pid_ioctl($pid, "get repc");

펄스출력모드 사용 예

펄스출력모드 사용(기본 펄스 출력)

```
<?php
$pid = pid_open("/mmap/st0");                // 0번 ST 열기
pid_ioctl($pid, "set div sec");              // 단위 설정: 초
pid_ioctl($pid, "set mode output pulse");    // 펄스출력모드 설정
pid_ioctl($pid, "set output dev uio0 0");    // 출력 핀 설정: uio0의 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);
?>
```

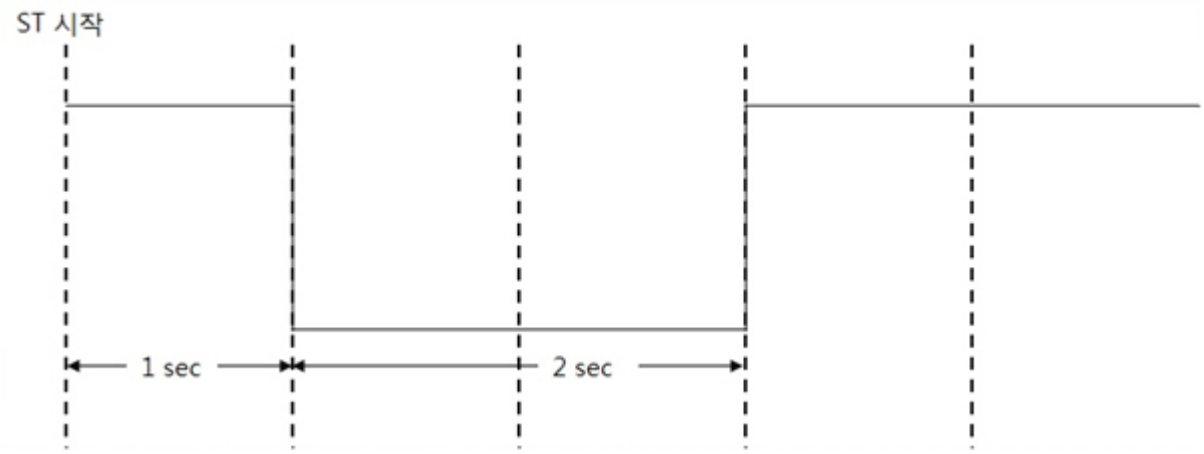
펄스출력모드의 출력은 기본적으로 low 신호에서 high신호로 변화됩니다. 각 상태의 유지시간은 설정 단위와 "set count"로 설정하는 T1과 T2에 의해 결정됩니다. 위 예제를 실행했을 때 ST출력 결과는 다음과 같습니다.



펄스출력모드 사용(반전 펄스 출력)

```
<?php
$pid = pid_open("/mmap/st0");               // 0번 ST 열기
pid_ioctl($pid, "set div sec");             // 단위 설정: 초
pid_ioctl($pid, "set mode output pulse");  // 펄스출력모드 설정
pid_ioctl($pid, "set output dev uio0 0");  // 출력 핀 설정: uio0의 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.

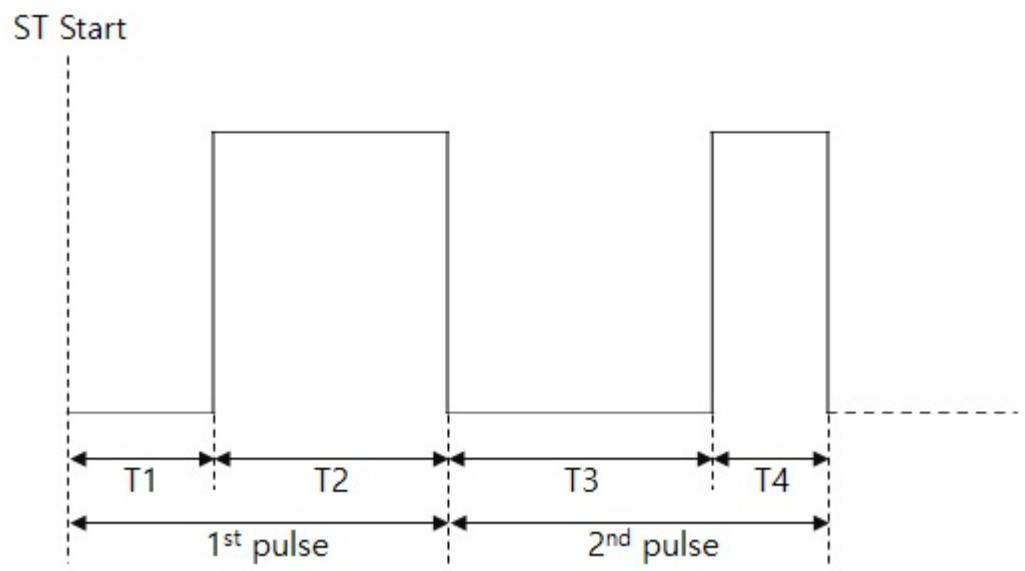


토글출력 vs 펄스출력

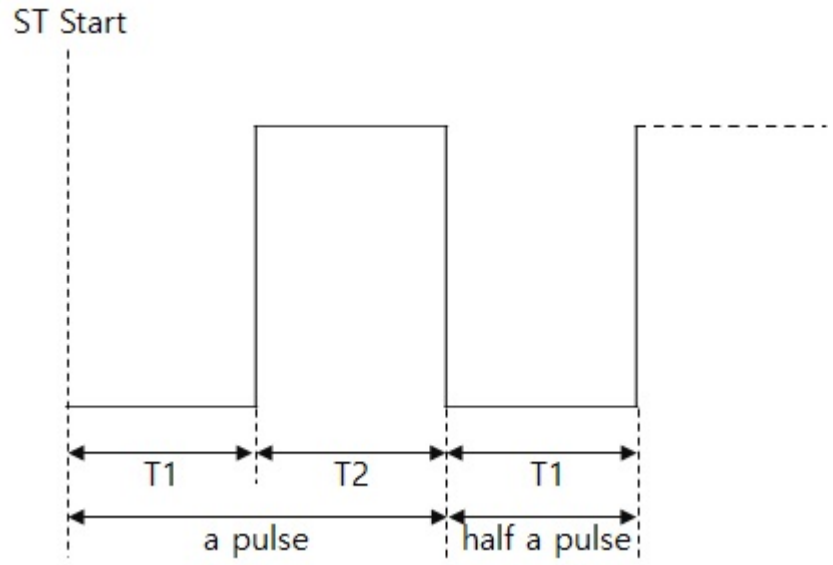
하나의 펄스 또는 반복되는 펄스를 생성하는 데에는 다음 두가지 경우가 있을 수 있습니다.

- 토글출력으로만 가능한 경우
- 토글출력 또는 펄스출력 모두 가능한 경우

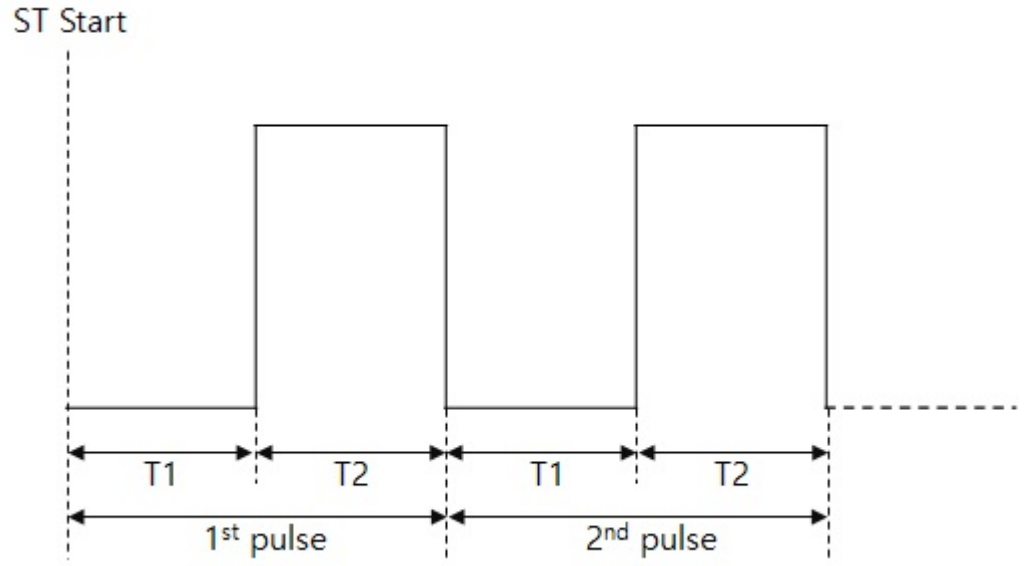
반복되는 펄스에서 펄스의 모양이 서로 다른경우에는 오직 토글출력만 사용할 수 있습니다.



반복되는 펄스에서 형태가 완전하지 못한 펄스가 포함되어 있는 경우에는 오직 토글출력만 사용할 수 있습니다.



반복되는 펄스에서 펄스의 모양이 모두 같은경우에는 토글출력 및 펄스출력 모두 사용할 수 있습니다.



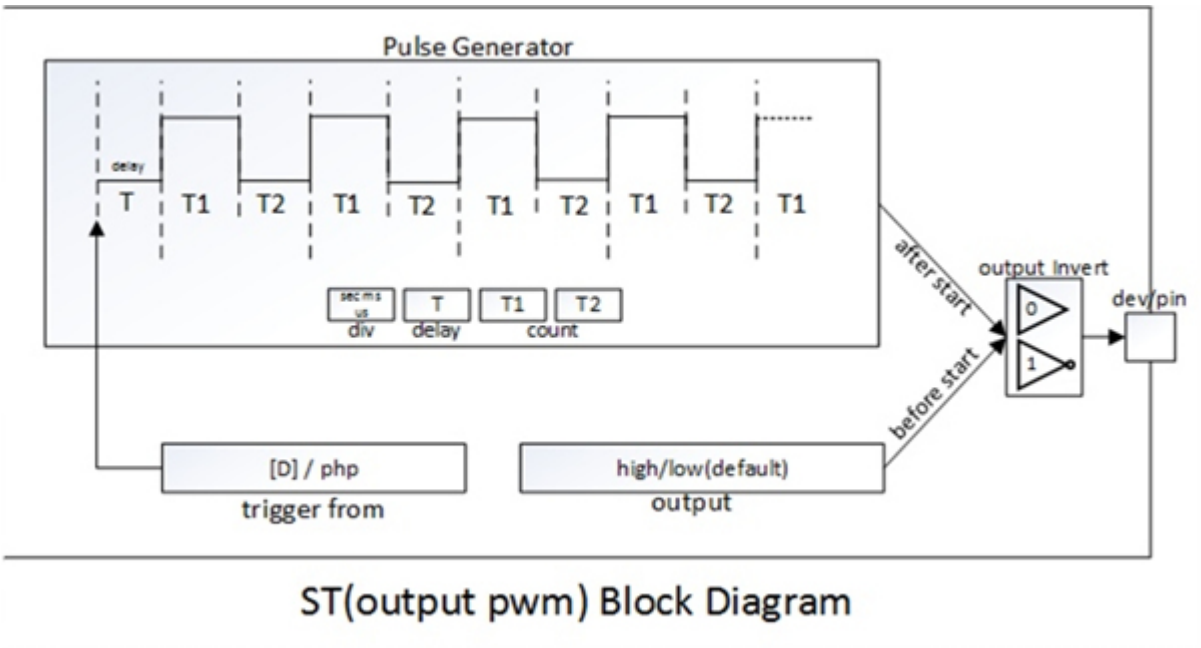
이 경우 두 모드의 소스코드는 반복횟수("set repc" 명령)를 설정하는 부분을 제외하고 거의 같습니다. 반복횟수 명령은 토글출력모드에서는 토글 횟수, 펄스출력모드에서는 펄스출력 횟수를 의미하므로, 같은 모양의 신호를 출력하기 위해서는 토글출력모드의 반복횟수가 펄스출력모드의 반복횟수의 2배가 되어야 합니다.

PWM출력모드

개요

이 모드는 PWM 신호를 정확하게 생성하는 데 사용됩니다. PWM 신호는 동일한 직사각형 펄스가 무한히 반복되는 형태입니다. PWM 신호는 주기적인 신호이며 각 사이클은 직사각형 펄스 입니다. PWM 신호는 low-level 신호와 high-level 신호로 구성됩니다.

다음은 ST 펄스출력모드의 블록다이어그램 입니다.



ST(output pwm) Block Diagram

명령어

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	od			오픈 드레인(Open-Drain)
		pp			푸쉬 풀(Push-Pull)
		low			LOW 출력
		high			HIGH 출력
		dev	uio0	#pin	출력 핀 설정: uio0의 #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

카운트 값 설정

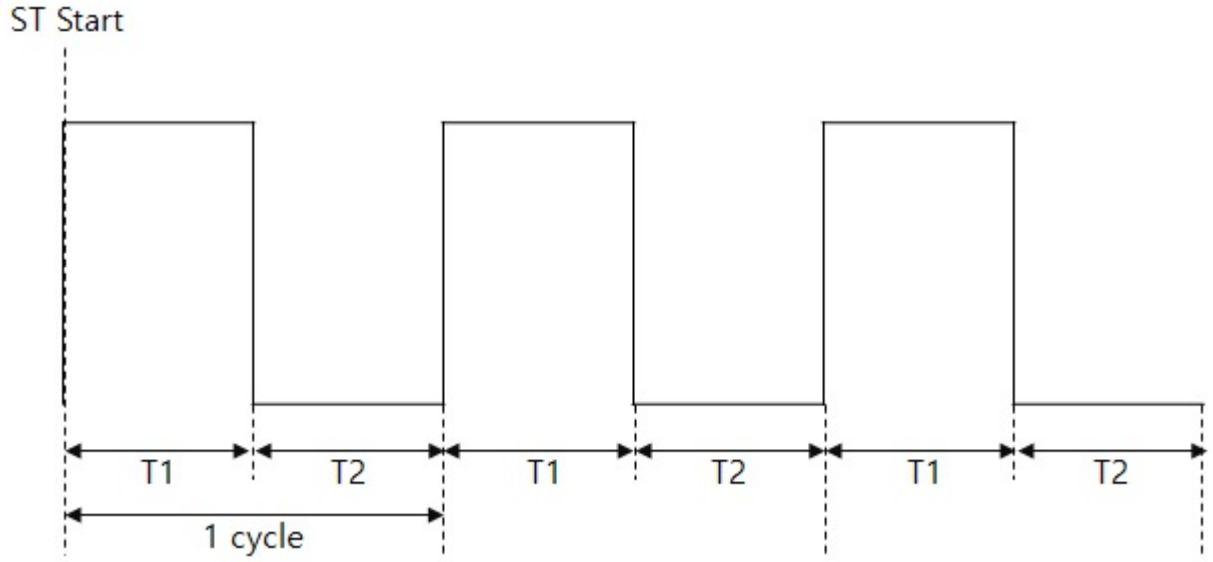
PWM출력모드에서는 이 명령어에 의해 직사각형 펄스의 low-level 및 high-level 유지시간을 설정할 수 있습니다. 카운트 설정 단위는 "set div"명령에 의한 단위가 사용됩니다.

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

PWM출력모드에서 설정 가능한 카운트 값 T1과 T2의 범위는 다음과 같습니다.

단위	T1, T2 설정 가능 범위(10μs ~ 30분)
마이크로 초	10 ~ 1,800,000,000
밀리 초	1 ~ 1,800,000
초	1 ~ 1,800

이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다. 아래 그림은 PWM출력모드의 출력 파형을 보여줍니다.



※ Duty cycle = T1 / (T1 + T2)
※ Frequency = 1 / (T1 + T2)

출력핀 설정

명령어	문법
set output dev D N	pid_ioctl($pid, "set output dev uio0 0");

ST의 출력모드를 사용하기 전에 이 명령을 이용하여 출력핀을 지정해야 합니다. 디바이스 이름(예: uio0)과 핀 번호를 D와 N에 각각 지정합니다.

출력상태 설정 [low/high]

이 명령은 ST 출력핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	ST 출력핀에 LOW출력

명령어	문법	설명
set output high	pid_ioctl($pid, "set output high");	ST 출력핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 ST 출력핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

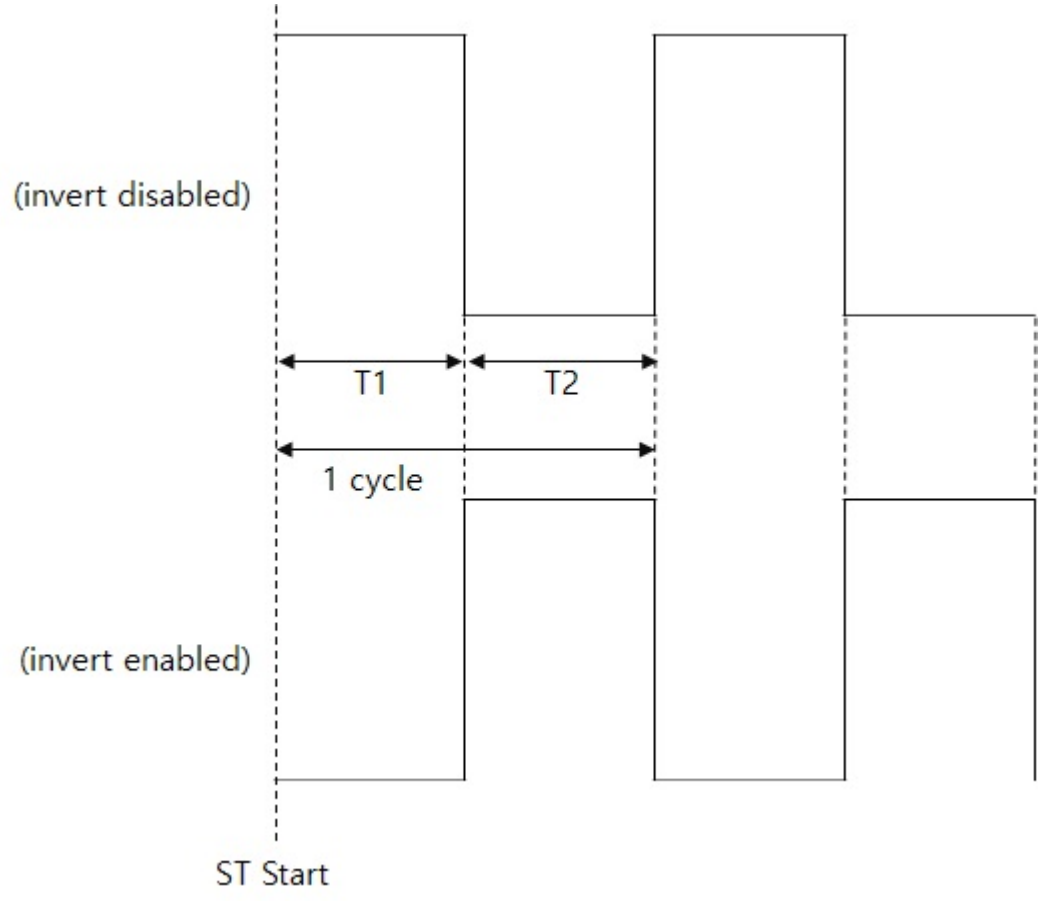
반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

반전출력이 활성화되어 있으면:

- 출력핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.



반전출력의 기본상태는 비활성화상태 입니다.

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");

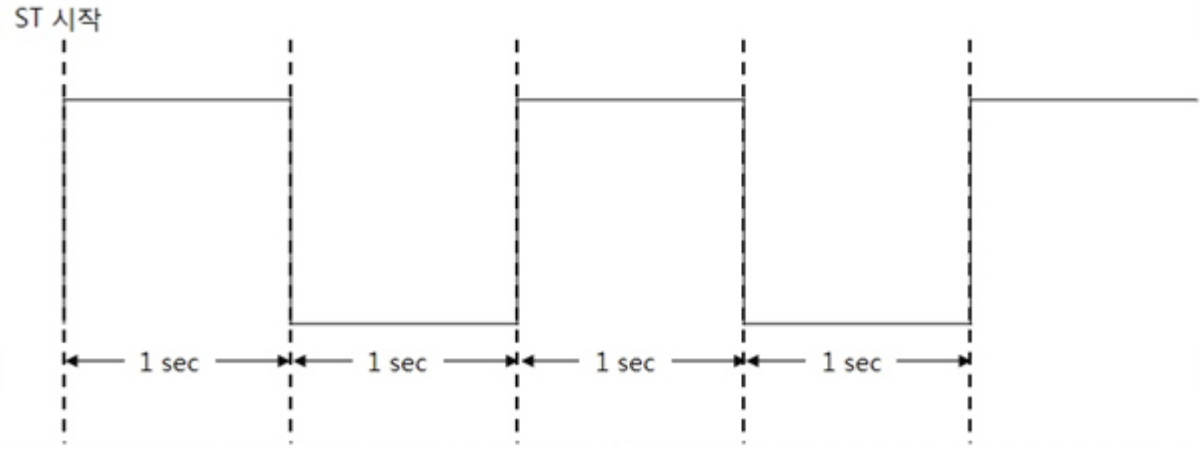
ST의 트리거 설정 기본 값은 "대상 없음(php)" 입니다.

PWM출력모드 사용 예

PWM출력모드 사용 예

```
<?php
$pid = pid_open("/mmap/st0");               // 0번 ST 열기
pid_ioctl($pid, "set div sec");             // 단위 설정: 초
pid_ioctl($pid, "set mode output pwm");     // PWM출력모드 설정
pid_ioctl($pid, "set output dev uio0 0");  // 출력 핀 설정: uio0의 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.



트리거

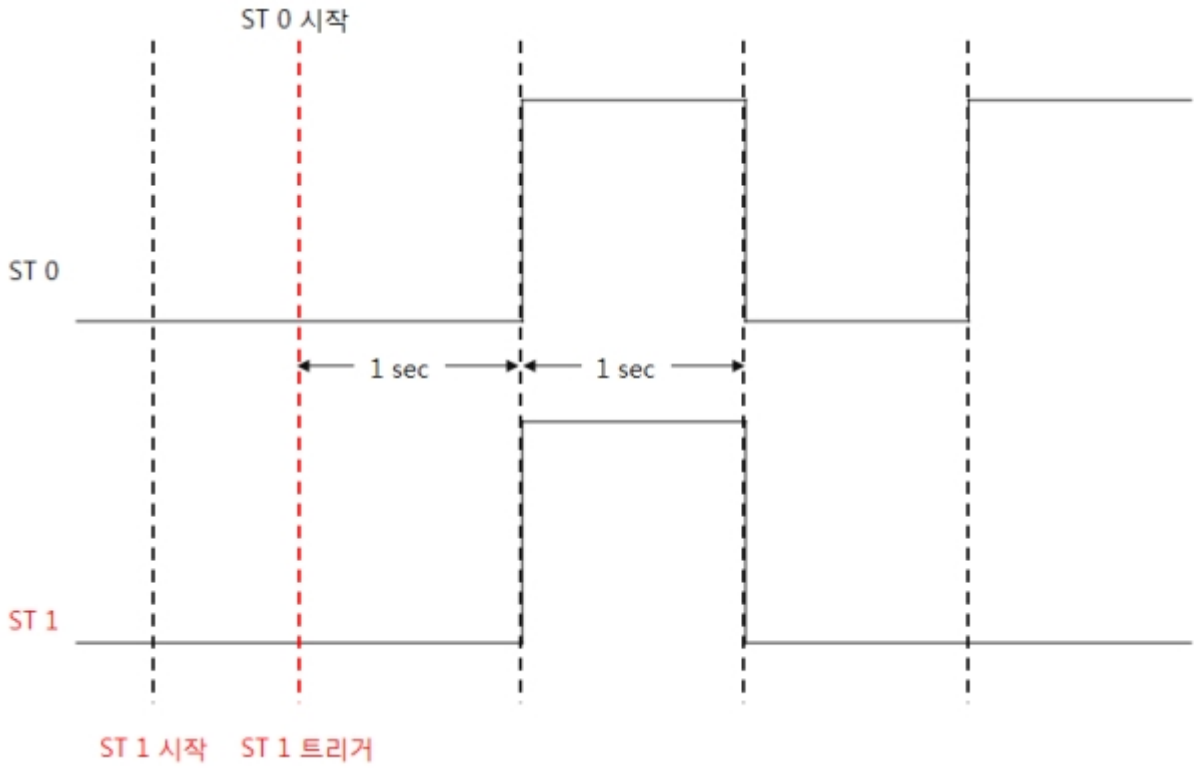
트리거 명령은 ST의 출력 시점을 또 다른 ST와 동기화시키고자 할 때 사용합니다. 아래 예제는 ST1의 출력 시점을 ST0와 동기화시키는 예 입니다.

트리거 사용 예

```
<?php
$pid0 = pid_open("/mmap/st0");              // 0번 ST 열기
pid_ioctl($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 uio0 0");  // 출력 핀 설정: uio0의 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 1");             // 출력 횟수 설정: 1
pid_ioctl($pid1, "set output dev uio0 1");  // 출력 핀 설정: uio0의 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.

출력 파형은 다음과 같습니다.



ST의 트리거 오차

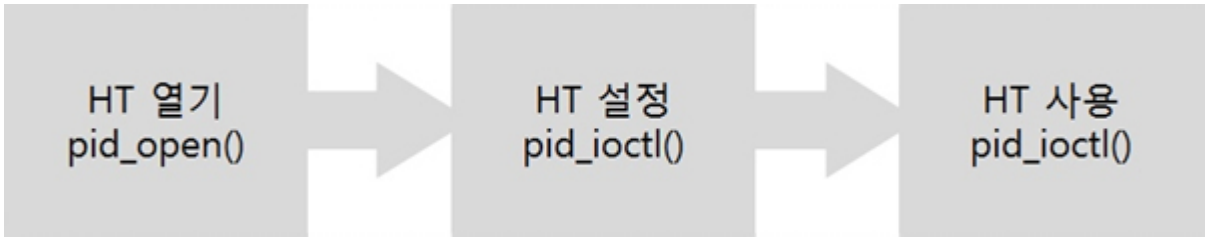
ST는 트리거 오차가 존재합니다. ST의 트리거 오차 범위는 다음과 같습니다.

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

※ 더 높은 정확도가 필요한 경우 HT를 사용하십시오.

HT 사용 절차

일반적인 HT 사용 절차는 다음과 같습니다.



HT 열기

HT를 열기 위해서는 pid_open함수를 사용합니다.

```
<?php
$pid = pid_open("/mmap/ht0");       // 0번 HT 열기
?>
```

※ 제품별 제공되는 HT 대한 정보는 부록을 참조하시기 바랍니다.

HT 설정 및 사용

HT는 다음 4가지 동작모드를 제공하며 HT를 설정하거나 사용하기 위해서는 각 모드에서 필요한 pid_ioctl함수를 사용해야 합니다.

모드	설명
토글출력모드	HT 핀으로 토글신호를 출력하기 위한 모드
펄스출력모드	HT 핀으로 펄스신호를 출력하기 위한 모드
PWM출력모드	무한 펄스출력모드
캡쳐모드	HT핀으로 들어오는 입력신호를 캡쳐하기 위한 모드

공통 명령어

다음 명령어들은 HT의 동작모드와 상관없이 공통적으로 사용됩니다.

명령어	하위 명령어			설명
set	mode	output	pulse	모드 설정: 펄스출력모드
			toggle	모드 설정: 토글출력모드
			pwm	모드 설정: PWM출력모드
		capture	rise	캡쳐모드 설정: 상승 에지
			fall	캡쳐모드 설정: 하강 에지
			toggle	캡쳐모드 설정: 상승 또는 하강 에지
	div	ms		단위 설정: 밀리 초
		us		단위 설정: 마이크로 초
	trigger	from	ht0	트리거 대상 설정 (ht0만 사용 가능)
			php	트리거 대상 설정: 없음
reset	-			초기화
get	state			상태 읽기
	div			분주 비 확인
	repc			남은 출력 및 캡쳐 횟수 확인
start	-			시작
stop	-			정지

모드 설정

HT는 다양한 출력 및 캡쳐모드를 지원합니다. 각각의 모드 설정 방법은 다음과 같습니다.

구분		문법
출력모드	펄스출력모드	pid_ioctl($pid, "set mode output pulse");
	토글출력모드	pid_ioctl($pid, "set mode output toggle");
	PWM출력모드	pid_ioctl($pid, "set mode output pwm");
캡쳐모드	상승 에지	pid_ioctl($pid, "set mode capture rise");
	하강 에지	pid_ioctl($pid, "set mode capture fall");
	상승/하강 에지	pid_ioctl($pid, "set mode capture toggle");

단위 설정

HT는 밀리 초 또는 마이크로 초 단위로 설정 가능하며 기본 값은 마이크로 초 입니다. 설정 방법은 다음과 같습니다.

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

트리거 설정

HT의 트리거 설정 방법은 다음과 같습니다.

구분	문법
ht0	pid_ioctl($pid, "set trigger from ht0"); // 트리거 대상: HT0
php	pid_ioctl($pid, "set trigger from php"); // 트리거 대상 없음

HT의 트리거의 대상으로는 HT0를 지정할 수 있습니다. HT의 트리거 설정 기본 값은 "대상 없음(php)" 입니다.

초기화

"reset"명령어는 다음과 같은 동작을 합니다:

- HT의 동작을 그 즉시 중단하고 초기화 합니다.
- HT핀에 LOW를 출력합니다.

구분	문법
초기화	pid_ioctl($pid, "reset");

상태 읽기

"get" 명령어는 HT의 여러 가지 상태를 읽는 명령어 입니다.

구분	문법
동작 상태	pid_ioctl($pid, "get state");
분주 비	pid_ioctl($pid, "get div");
남은 출력 / 캡쳐 횟수	pid_ioctl($pid, "get repc");

각 상태 확인에 대한 반환 값은 정수 형태이며 다음과 같습니다.

구분	반환 값	설명
디바이스 상태	0	정지
	1 ~ 3	동작 중
분주 비	42	마이크로 초
	42000	밀리 초
남은 출력 / 캡쳐 횟수	0 ~ 64	-

시작

HT를 시작시키기 위해서는 "start"명령을 사용합니다.

구분	문법
시작	pid_ioctl($pid, "start");

정지

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

구분	문법
정지	pid_ioctl($pid, "stop");

토글출력모드

개요

이 모드는 다양한 지속시간을 갖는 하나 또는 여러개의 직사각형 펄스를 출력하는데 사용합니다. 이 모드에서 HT핀은 미리 정의된 시간 간격에 따라 출력이 토글됩니다.

HT의 출력파형은 다음에 따라 달라집니다:

- 타이머가 시작되는 시점의 HT핀의 상태
- 두 개의 연속적인 토글출력 사이의 지속시간(카운트 값)
- 반복 횟수

명령어

명령어	하위 명령어			설명
set	mode	output	toggle	모드 설정: 토글출력모드
	div	ms		단위 설정: 밀리 초
		us		단위 설정: 마이크로 초
	output	od		오픈 드레인(Open-Drain)
		pp		푸쉬 풀(Push-Pull)
		low		LOW 출력
		high		HIGH 출력
		invert	0	정상(비 반전)출력
			1	반전 출력
	count	[T1] ... [T8]		출력 타이밍 설정
	repc	[N]		출력 횟수 설정
	trigger	from	ht0	트리거 대상 설정: ht0만 가능
			php	트리거 대상 설정: 없음
reset	-			초기화
get	state			상태 읽기
	div			분주 비 읽기
	repc			남은 출력 횟수 읽기
start	-			시작
stop	-			정지

반복횟수 설정

토글출력모드에서 이 명령어는 토글 신호의 반복횟수를 설정합니다.

명령어	문법	N의 범위
set repc	pid_ioctl($pid, "set repc N");	0 ~ 64

반복횟수의 기본값은 0입니다. 이 값이 0이면 반복횟수는 최대 반복횟수(64)로 설정됩니다.
(카운트 값 설정의 예제 파형 참조)

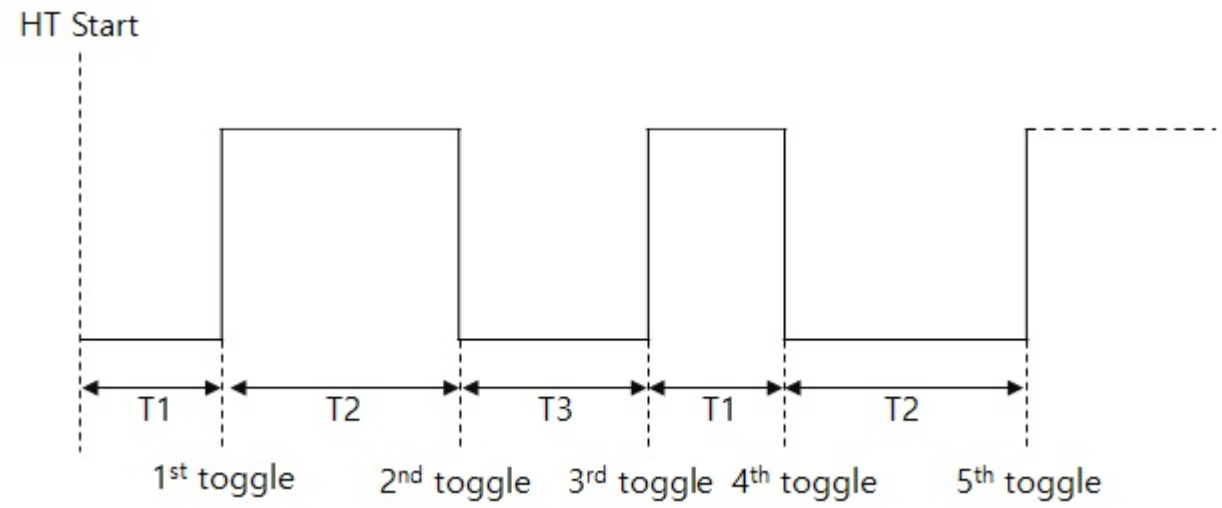
카운트 값 설정

토글출력모드에서 이 설정은 HT핀의 토글출력 타이밍을 지정하는 데 사용됩니다. 카운트 값의 설정 단위는 "set div"명령에 의한 단위가 사용됩니다. 카운트 값은 최소 1개부터 최대 8개까지 설정할 수 있습니다. 각 카운트 값은 1 에서 32764 사이에서 설정합니다.

명령어	문법	T의 범위
토글출력모드	pid_ioctl($pid, "set count T1 T2 ... T8");	1 ~ 32764

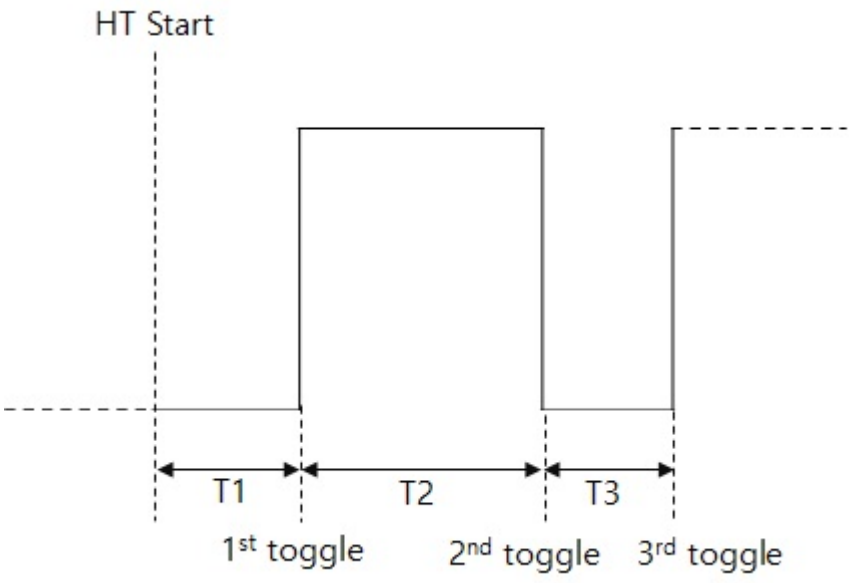
이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다. 토글출력모드에서 카운트 값을 2개 이상 설정하면 이 값들은 출력 시점마다 순서대로 사용됩니다. 만약 반복횟수가 설정한 카운트 값의 수 보다 많으면 다시 첫 번째 값부터 순서대로 사용됩니다. 아래 그림은 다음 조건하에서의 출력 파형을 보여줍니다.

- "set count T1 T2 T3": 3개(T1, T2, T3)의 카운트 값 설정
- "set repc 5": 5번의 반복횟수 설정
- 타이머 시작 시점의 HT출력핀의 상태는 LOW

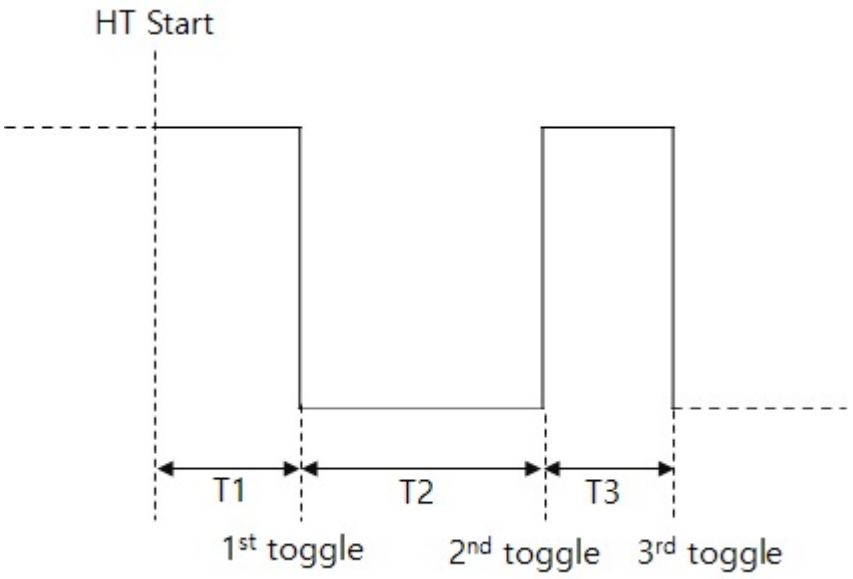


출력 파형은 타이머가 시작되는 시점의 HT 출력핀의 상태에 따라 달라집니다. 다음 예는 HT 출력핀의 상태가 다를 때의 파형을 보여줍니다. 반복 횟수는 3회이고 카운트 값은 각각 T1, T2 그리고 T3입니다.

- 타이머가 시작되는 시점에 HT 출력핀의 상태가 LOW일 때



- 타이머가 시작되는 시점에 HT 출력핀의 상태가 HIGH일 때



HT 출력핀의 상태는:

- 시스템 리부트 이후에 LOW
- "reset" 명령 이후에 LOW
- "set output low"명령 직후에 LOW
- "set output high"명령 직후에 HIGH
- 타이머 출력 모드(토글, 펄스 또는 PWM)에 따라 LOW 또는 HIGH

두개의 연속적인 토글 출력 사이의 지속시간(카운트 값)은 "set count"명령으로 설정합니다.

출력상태 설정 [low/high]

이 명령은 HT핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	HT핀에 LOW출력
set output high	pid_ioctl($pid, "set output high");	HT핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 HT핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

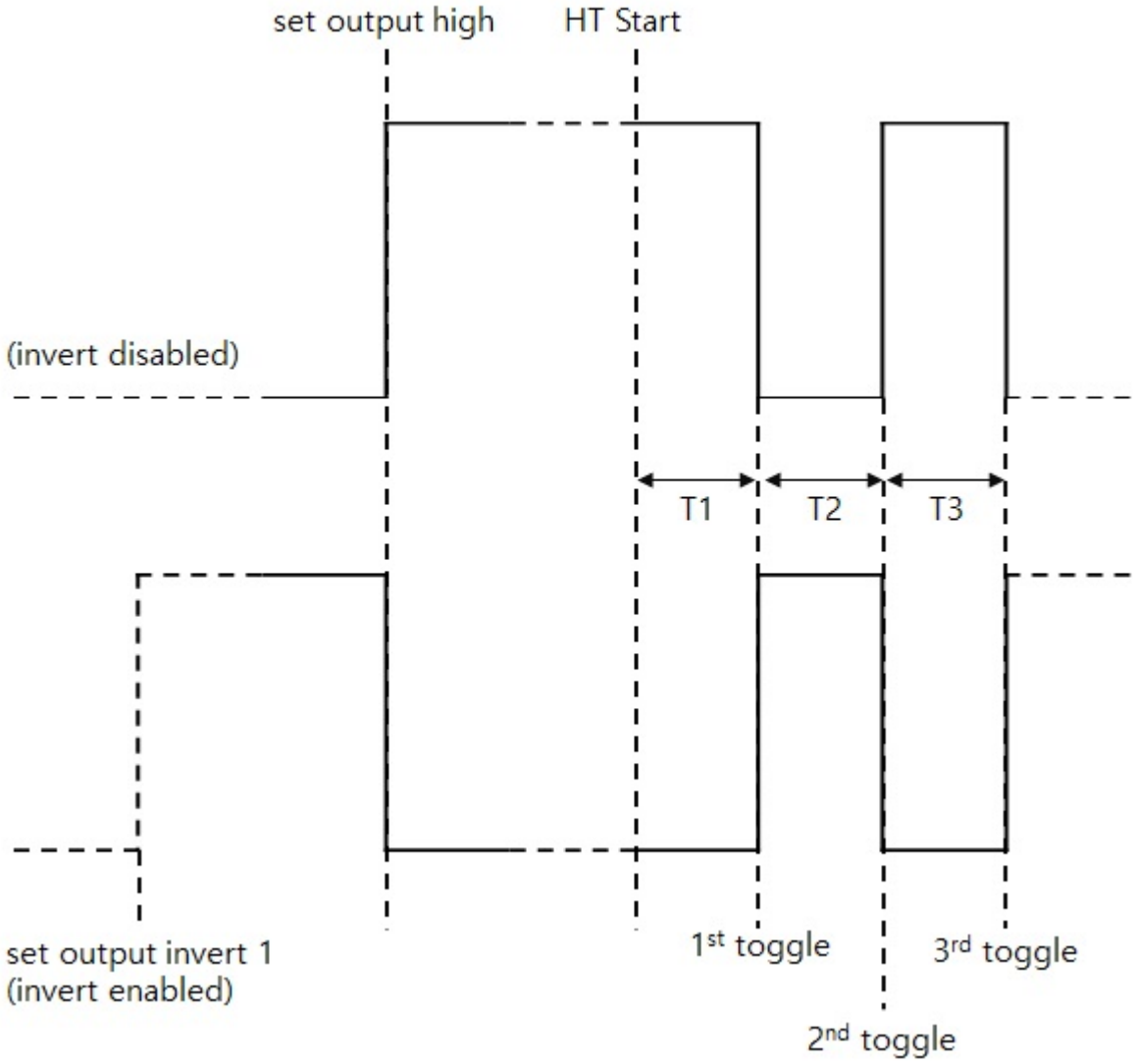
반전출력이 활성화되어 있으면:

- HT핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.

반전출력 활성상태가 바뀌면(반전출력 활성화 또는 비활성화) HT핀의 상태는 즉시 토글됩니다.

반전출력의 기본상태는 비활성화상태 입니다.

다음은 반전출력모드의 활성화/비활성화에 따른 파형의 차이를 보여줍니다. 반복 횟수는 3회이고 카운트 값은 각각 T1, T2 그리고 T3입니다. 타이머 시작 전 "set output high"명령이 사용되었습니다.



위 그림에서 보는것과 같이 반전출력이 활성화되면 "set output high"명령에 의한 출력이 LOW가 됩니다.

토글출력모드 사용 예

토글출력모드 사용

```
<?php
$pid = pid_open("/mmap/ht0");              // 0번 HT 열기
```

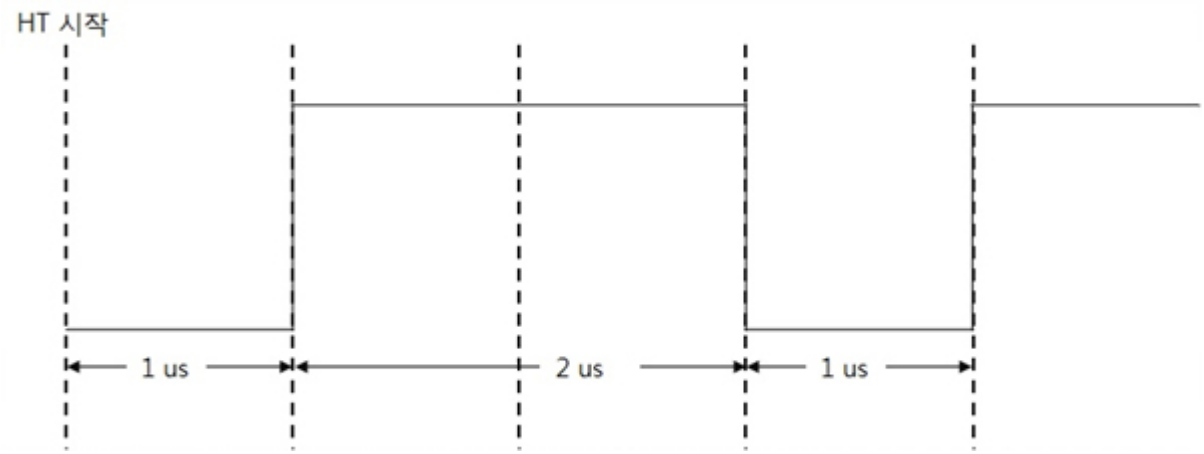
```
pid_ioctl($pid, "set div us");                // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode output toggle");  // 토글출력모드 설정
pid_ioctl($pid, "set repc 1");                // 출력 횟수 설정: 1
pid_ioctl($pid, "set count 1");               // T1 설정: 1
pid_ioctl($pid, "start");                     // HT 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
?>
```

토글출력모드에서의 "set count"는 HT의 시작시점부터 출력을 내보내는 시점까지의 시간을 의미합니다. 위 예제를 실행했을 때 HT의 출력 결과는 다음과 같습니다.

반복적 토글출력모드

```
<?php
$pid = pid_open("/mmap/ht0");                // 0번 HT 열기
pid_ioctl($pid, "set div us");                // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode output toggle");  // 토글출력모드 설정
pid_ioctl($pid, "set repc 3");                // 출력 횟수 설정: 3회
pid_ioctl($pid, "set count 1 2 1");          // 카운트 값 설정: 1, 2, 1
pid_ioctl($pid, "start");                     // HT 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
?>
```

위 예제에서는"set repc"명령어로 토글출력 횟수를 3회로 설정하고 토글출력 타이밍을 위해 "set count"로 T1, T2 및 T3을 각각 1, 2 그리고 1마이크로 초로 설정했습니다. 위 예제를 실행했을 때 HT출력 결과는 다음과 같습니다.


HT 시작
1 us
2 us
1 us

펄스출력모드

개요

이 모드는 하나 또는 여러개의 직사각형 펄스를 출력하는데 사용합니다. 이 모드에서 각 직사각형 펄스의 모양은 모두 동일합니다.

HT의 출력파형은 다음에 따라 달라집니다:

- 타이머가 시작되는 시점의 HT핀의 상태와는 무관
- low-level과 high-level의 지속시간(카운트 값)
- 반복 횟수

명령어

명령어	하위 명령어			설명
set	mode	output	pulse	모드 설정: 펄스출력모드
	div	ms		단위 설정: 밀리 초
		us		단위 설정: 마이크로 초
	output	od		오픈 드레인(Open-Drain)
		pp		푸쉬 풀(Push-Pull)
		low		LOW 출력
		high		HIGH 출력
		invert	0	정상(비 반전)출력
			1	반전 출력
	count	[T1] [T2]		출력 타이밍 설정
	repc	[N]		출력 횟수 설정
	trigger	from	ht0	트리거 대상 설정: ht0만 가능
			php	트리거 대상 설정: 없음
reset	-			초기화
get	state			상태 읽기
	div			분주 비 읽기
	repc			남은 출력 횟수 읽기
start	-			시작
stop	-			정지

반복횟수 설정

펄스출력모드에서는 출력 신호의 반복횟수를 설정할 수 있습니다.

명령어	문법	N의 범위
set repc	pid_ioctl($pid, "set repc N");	0 ~ 64

반복횟수의 기본값은 0입니다. 이 값이 0이면 반복횟수는 최대 반복횟수(64)로 설정됩니다.
(카운트 값 설정의 예제 파형 참조)

카운트 값 설정

하나의 직사각형 펄스는 high-level 신호와 low-level 신호로 구성됩니다. 펄스출력모드에서는 이 명령어에 의해 직사각형 펄스의 low-level 및 high-level 유지시간을 설정할 수 있습니다. 카운트 값의 설정 단위는 "set div"명령에 의한 단위가 사용됩니다.

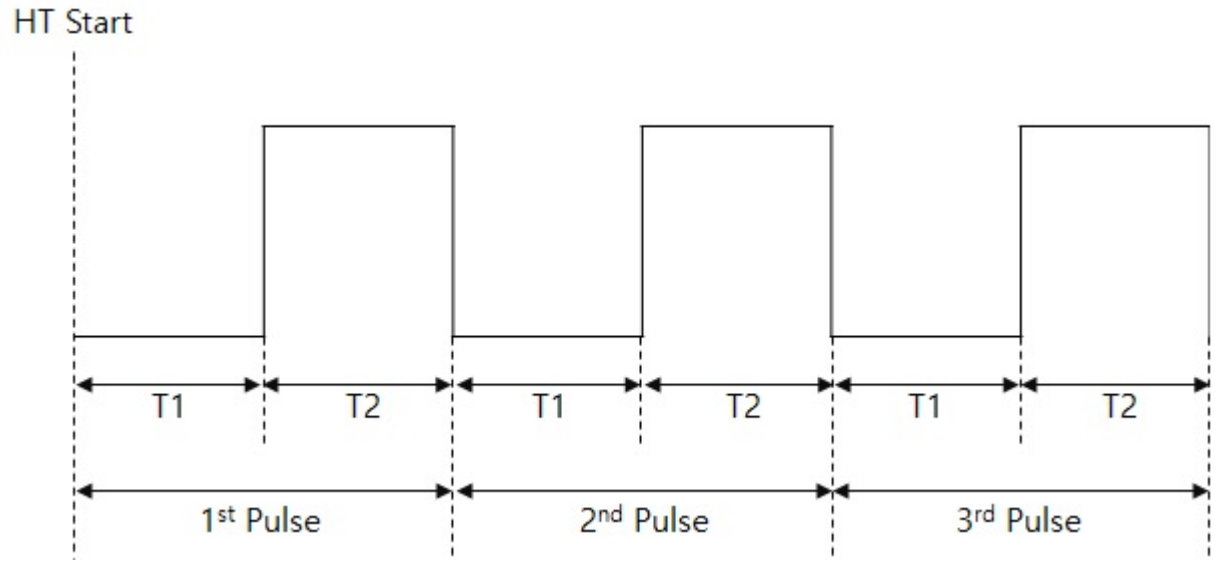
명령어	문법
set count	pid_ioctl($pid, "set count T1 T2");

펄스출력모드에서 설정 가능한 카운트 값 T1과 T2의 범위는 다음과 같습니다.

구분	카운트 값의 범위
T1	1 ~ 32763
T2	1 ~ 32763
T1 + T2	2 ~ 32764

이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다.
아래 그림은 다음 조건하에서의 출력 파형을 보여줍니다.

- "set repc 3": 3번의 반복횟수 설정
- "set count T1 T2": 카운트 값 T1 및 T2 설정



출력상태 설정 [low/high]

이 명령은 HT핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	HT핀에 LOW출력
set output high	pid_ioctl($pid, "set output high");	HT핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 HT핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

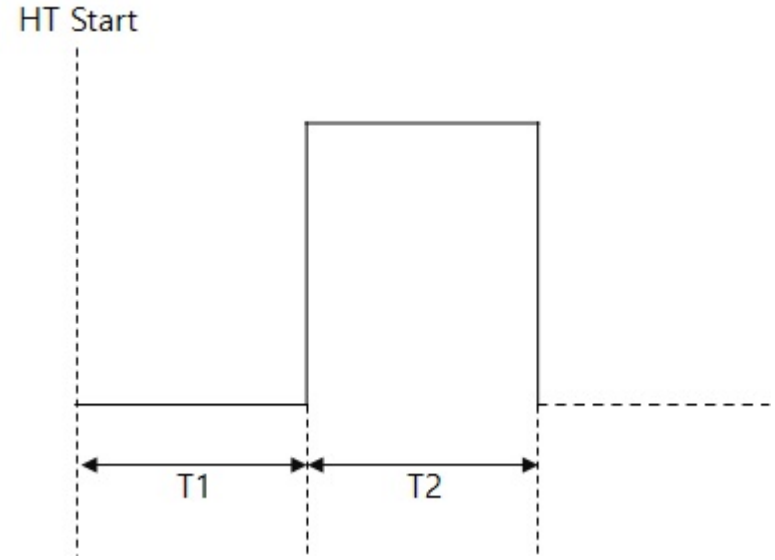
명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

반전출력이 활성화되어 있으면:

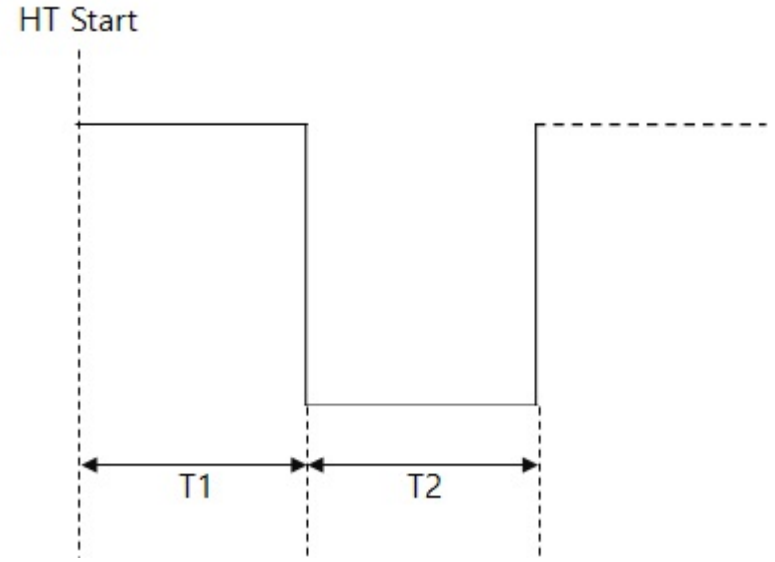
- HT핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.

반전출력 활성상태가 바뀌면(반전출력 활성화 또는 비활성화) HT 출력핀의 상태는 즉시 토글됩니다. 즉, 이 명령어에 의해 펄스의 형태가 결정됩니다. 펄스는 다음 두가지의 형태가 있습니다.

- 반전 출력 비활성화



- 반전 출력 활성화



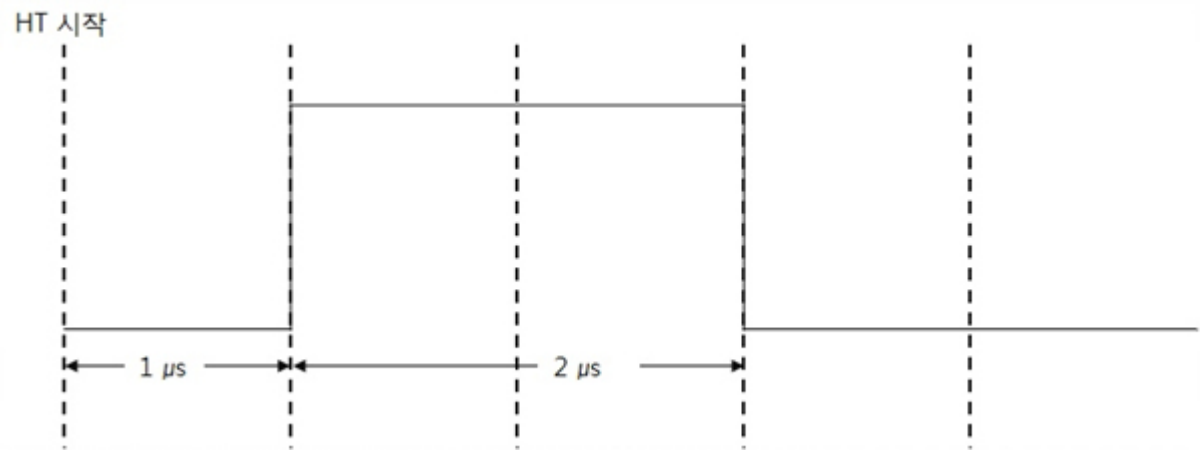
반전출력의 기본상태는 비활성화상태 입니다.

펄스출력모드 사용 예

펄스출력모드 사용(기본 펄스 출력)

```
<?php
$pid = pid_open("/mmap/ht0");                // 0번 HT 열기
pid_ioctl($pid, "set div us");               // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode output pulse");    // 펄스출력모드 설정
pid_ioctl($pid, "set count 1 2");            // 카운트 값 설정: 1, 2
pid_ioctl($pid, "set repc 1");               // 출력 횟수 설정: 1
pid_ioctl($pid, "start");                    // HT 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
?>
```

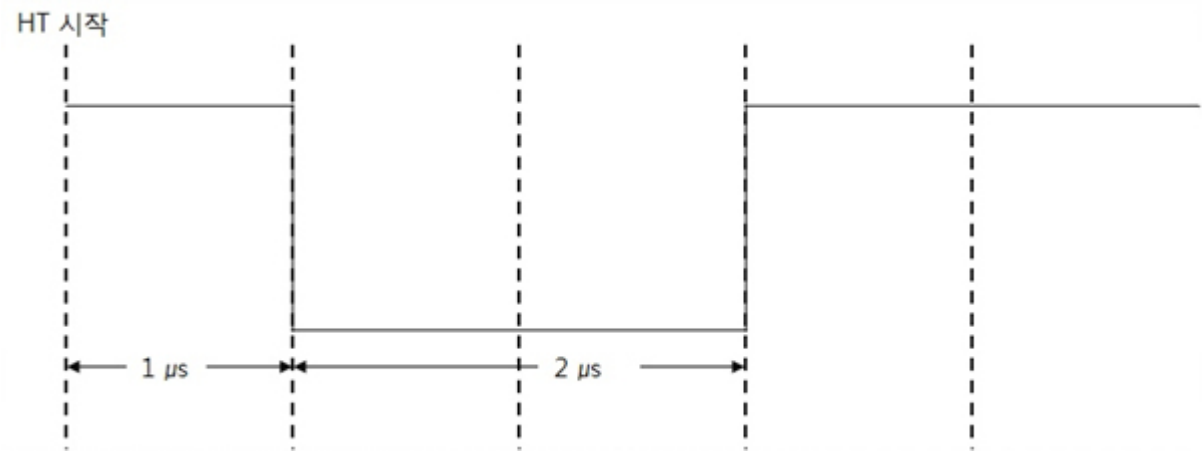
펄스출력모드의 출력은 기본적으로 low 신호에서 high신호로 변화됩니다. 각 상태의 유지시간은 설정 단위와 "set count"로 설정하는 T1과 T2에 의해 결정됩니다. 위 예제를 실행했을 때 HT핀의 출력 결과는 다음과 같습니다.



펄스출력모드 사용(반전 펄스 출력)

```
<?php
$pid = pid_open("/mmap/ht0");                // 0번 HT 열기
pid_ioctl($pid, "set div us");               // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode output pulse");    // 펄스출력모드 설정
pid_ioctl($pid, "set count 1 2");            // 카운트 값 설정: 1, 2
pid_ioctl($pid, "set repc 1");               // 출력 횟수 설정: 1
pid_ioctl($pid, "set output invert 1");      // 출력 반전
pid_ioctl($pid, "start");                    // HT 시작
while(pid_ioctl($pid, "get state"));
pid_close($pid);
?>
```

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

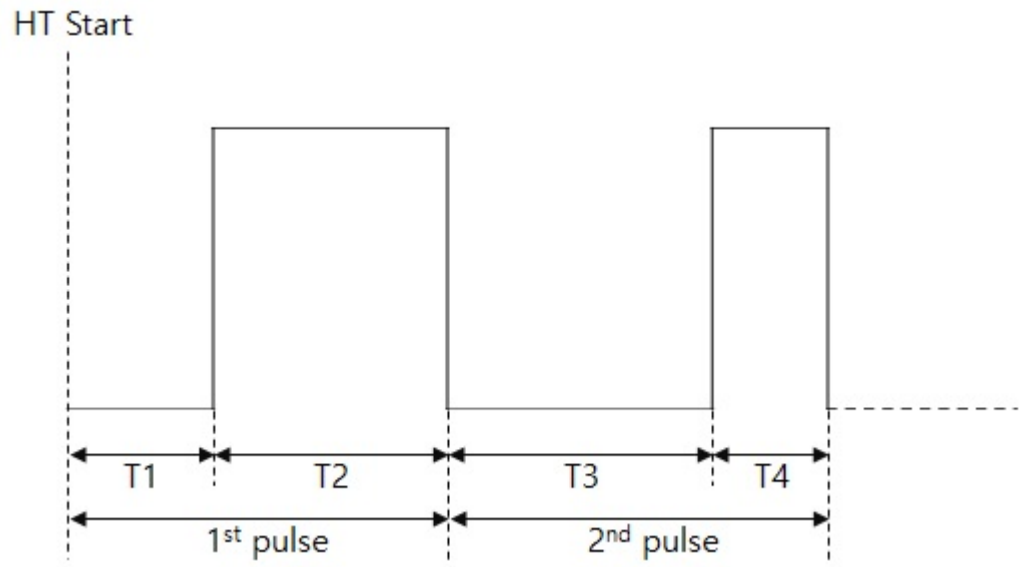

HT 시작
1 μs
2 μs

토글출력 vs 펄스출력

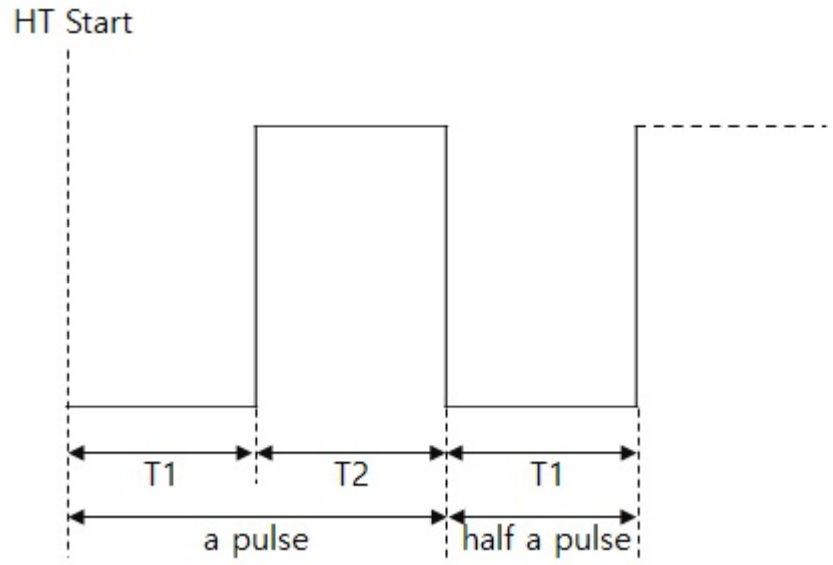
하나의 펄스 또는 반복되는 펄스를 생성하는 데에는 다음 두가지 경우가 있을 수 있습니다.

- 토글출력으로만 가능한 경우
- 토글출력 또는 펄스출력 모두 가능한 경우

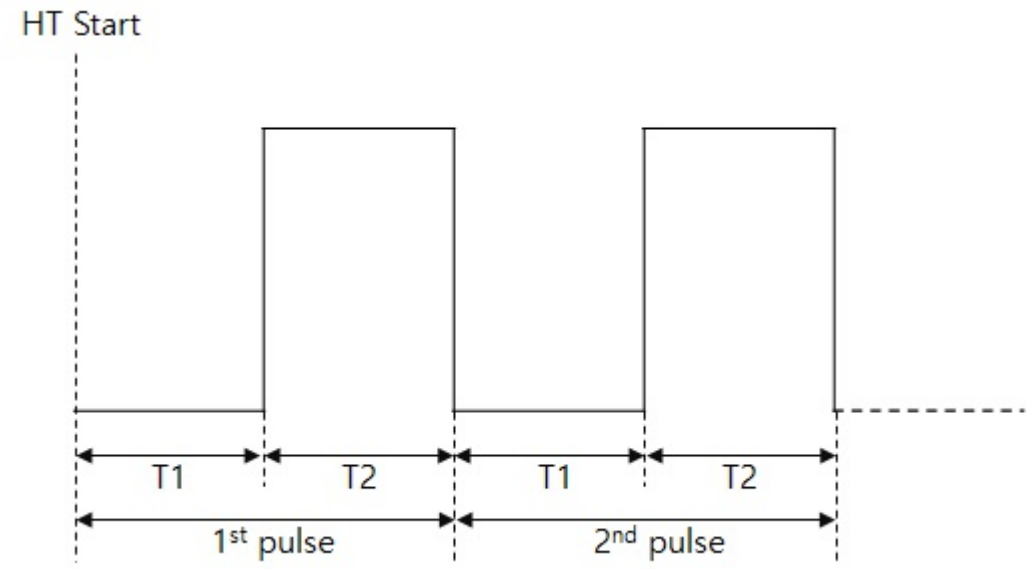
반복되는 펄스에서 펄스의 모양이 서로 다른경우에는 오직 토글출력만 사용할 수 있습니다.



반복되는 펄스에서 형태가 완전하지 못한 펄스가 포함되어 있는 경우에는 오직 토글출력만 사용할 수 있습니다.



반복되는 펄스에서 펄스의 모양이 모두 같은경우에는 토글출력 및 펄스출력 모두 사용할 수 있습니다.



이 경우 두 모드의 소스코드는 반복횟수("set repc" 명령)를 설정하는 부분을 제외하고 거의 같습니다. 반복횟수 명령은 토글출력모드에서는 토글 횟수, 펄스출력모드에서는 펄스출력 횟수를 의미하므로, 같은 모양의 신호를 출력하기 위해서는 토글출력모드의 반복횟수가 펄스출력모드의 반복횟수의 2배가 되어야 합니다.

PWM출력모드

개요

이 모드는 PWM 신호를 정확하게 생성하는 데 사용됩니다. PWM 신호는 동일한 직사각형 펄스가 무한히 반복되는 형태입니다. PWM 신호는 주기적인 신호이며 각 사이클은 직사각형 펄스 입니다. PWM 신호는 low-level 신호와 high-level 신호로 구성됩니다.

명령어

명령어	하위 명령어			설명
set	mode	output	pwm	모드 설정: PWM출력모드
	div	ms		단위 설정: 밀리 초
		us		단위 설정: 마이크로 초
	output	od		오픈 드레인(Open-Drain)
		pp		푸쉬 풀(Push-Pull)
		low		LOW 출력
		high		HIGH 출력
		invert	0	정상(비 반전)출력
			1	반전 출력
	count	[T1] [T2]		출력 타이밍 설정
	trigger	from	ht0	트리거 대상 설정: ht0만 가능
			php	트리거 대상 설정: 없음
reset	-			초기화
get	state			상태 읽기
	div			분주 비 읽기
start	-			시작
stop	-			정지

카운트 값 설정

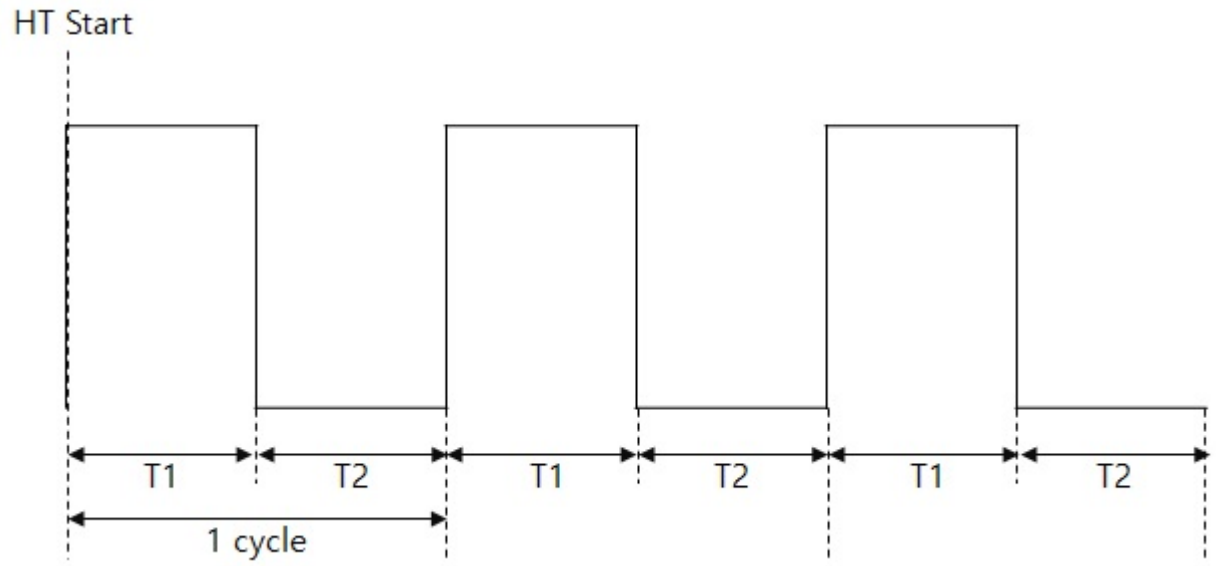
PWM출력모드에서는 이 명령어에 의해 직사각형 펄스의 low-level 및 high-level 유지시간을 설정할 수 있습니다. 카운트 설정 단위는 "set div"명령에 의한 단위가 사용됩니다.

명령어	문법
set count	pid_ioctl($pid, "set count T1 T2");

PWM출력모드에서 설정 가능한 카운트 값 T1과 T2의 범위는 다음과 같습니다.

구분	카운트 값의 범위
T1	1 ~ 32763
T2	1 ~ 32763
T1 + T2	2 ~ 32764

이 명령은 타이머 시작 전에 설정되어야 합니다. 그렇지 않은경우 에러가 발생합니다. 아래 그림은 PWM출력모드의 출력 파형을 보여줍니다.



※ Duty cycle = T1 / (T1 + T2)
※ Frequency = 1 / (T1 + T2)

출력상태 설정 [low/high]

이 명령은 HT핀에 즉시 LOW 또는 HIGH를 출력합니다.

명령어	문법	설명
set output low	pid_ioctl($pid, "set output low");	HT핀에 LOW출력
set output high	pid_ioctl($pid, "set output high");	HT핀에 HIGH출력

반전출력이 활성화되어 있으면 이 명령에 의한 출력 또한 반전되어 출력됩니다.

출력타입 설정 [od/pp]

이 명령은 HT핀의 출력타입을 설정합니다.

명령어	문법	설명
set output pp	pid_ioctl($pid, "set output pp");	출력타입을 푸쉬-풀로 설정
set output od	pid_ioctl($pid, "set output od");	출력타입을 오픈-드레인으로 설정

출력타입의 기본값은 푸쉬-풀입니다.

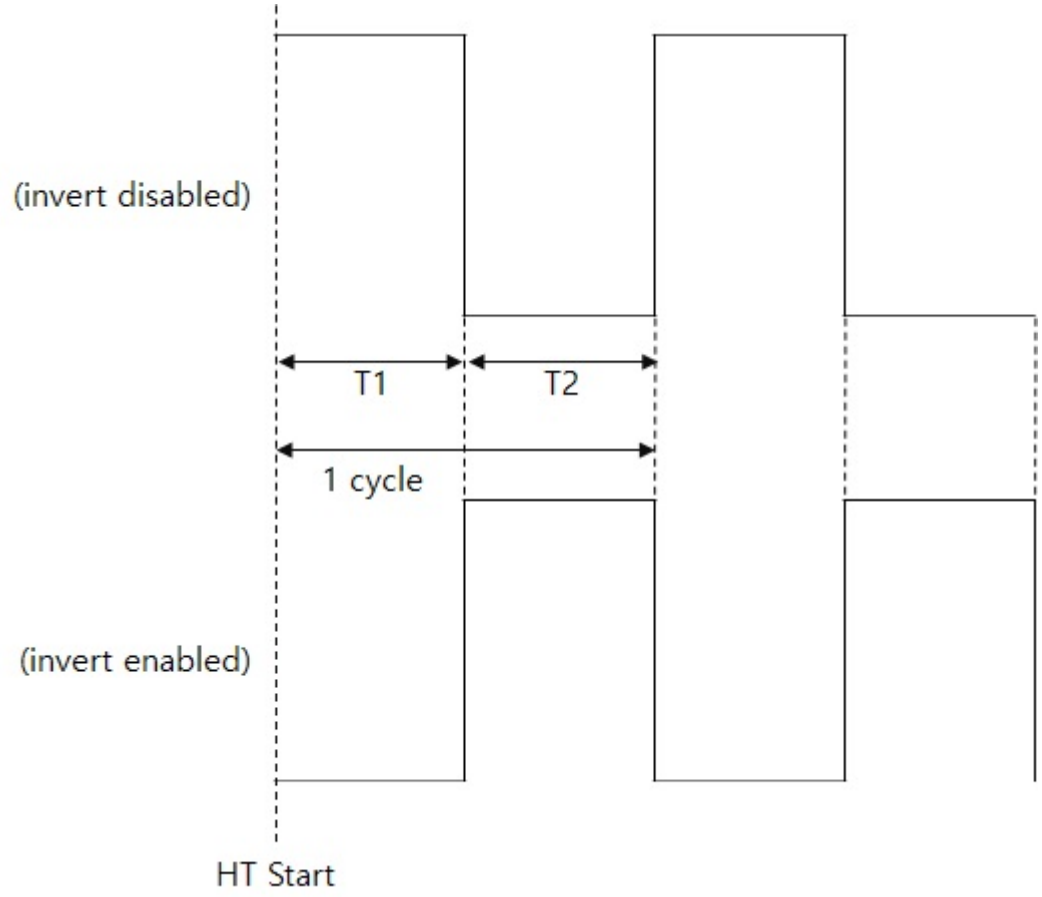
반전출력 설정 [invert 0/1]

이 명령은 반전출력의 활성화/비활성화를 설정합니다.

명령어	문법	설명
set output invert 0	pid_ioctl($pid, "set output invert 0");	반전출력 비활성화
set output invert 1	pid_ioctl($pid, "set output invert 1");	반전출력 활성화

반전출력이 활성화되어 있으면:

- HT핀의 출력신호가 정상동작과는 반대로 출력됩니다.
- "set output high" 및 "set output low"명령어에 의한 출력 또한 반전되어 출력됩니다.



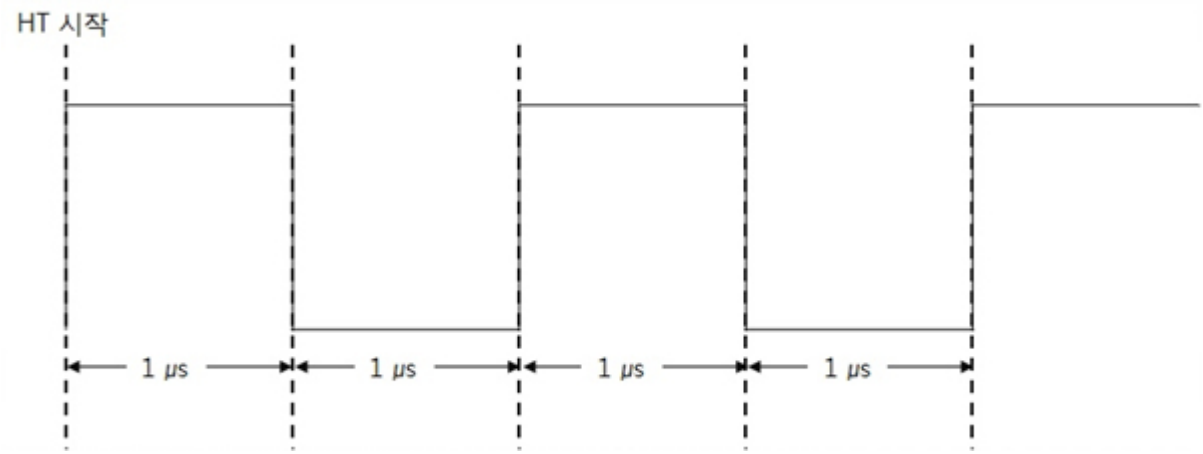
반전출력의 기본상태는 비활성화상태 입니다.

PWM출력모드 사용 예

PWM출력모드 사용 예

```
<?php
$pid = pid_open("/mmap/ht0");              // 0번 HT 열기
pid_ioctl($pid, "set div us");             // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode output pwm");    // PWM출력모드 설정
pid_ioctl($pid, "set count 1 1");          // 카운트 값 설정: 1, 1
pid_ioctl($pid, "start");                  // HT 시작
usleep(50);
pid_ioctl($pid, "stop");                   // HT 정지
pid_close($pid);
?>
```

위 예제를 실행했을 때 HT의 출력 결과는 다음과 같습니다.


HT 시작
1 μs
1 μs
1 μs
1 μs

출력모드와 트리거

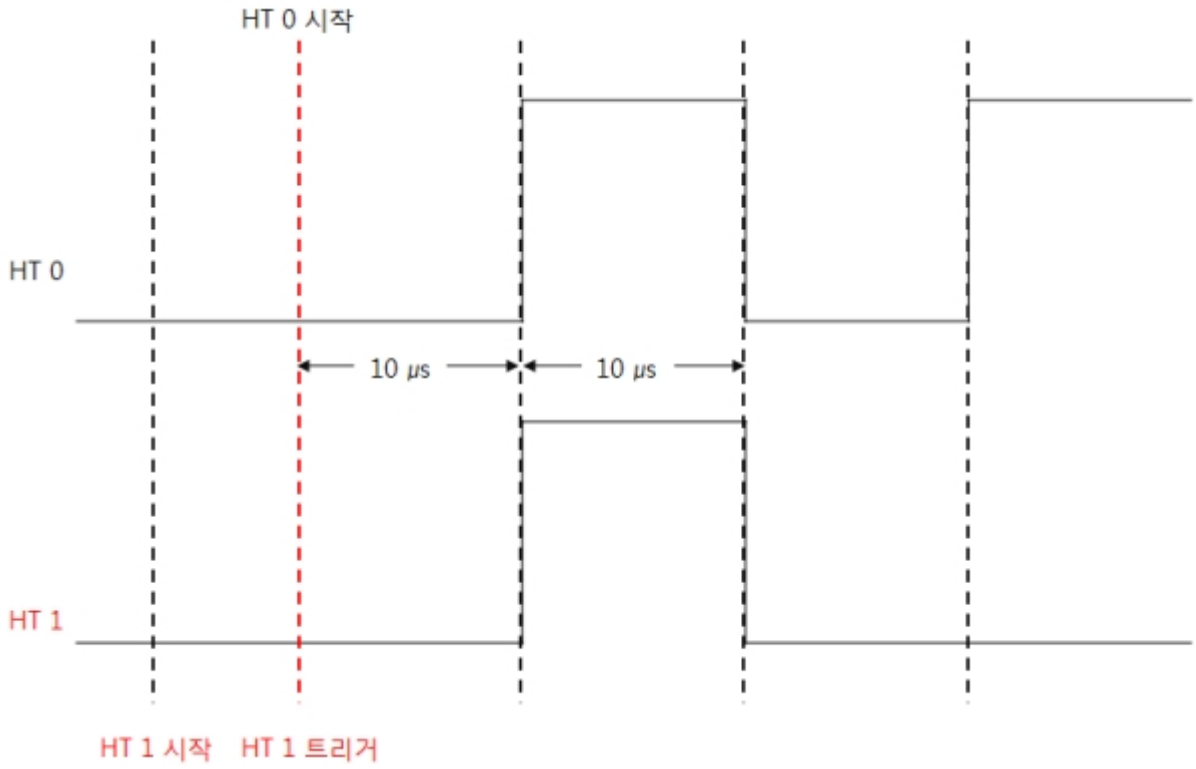
출력모드에서 트리거 명령은 HT의 출력 시점을 HT0와 동기화시키고자 할 때 사용합니다. 아래 예제는 HT1의 출력 시점을 HT0와 동기화시키는 예 입니다.

출력모드에서 트리거 사용 예

```
<?php
$pid0 = pid_open("/mmap/ht0");              // 0번 HT 열기
pid_ioctl($pid0, "set div us");             // 단위 설정: 마이크로 초
pid_ioctl($pid0, "set mode output pulse");  // 펄스출력모드 설정
pid_ioctl($pid0, "set count 10 10");        // 카운트 값 설정: 10, 10
pid_ioctl($pid0, "set repc 2");             // 출력 횟수 설정: 2

$pid1 = pid_open("/mmap/ht1");              // 1번 HT 열기
pid_ioctl($pid1, "set div us");             // 단위 설정: 마이크로 초
pid_ioctl($pid1, "set mode output pulse");  // 펄스출력모드 설정
pid_ioctl($pid1, "set trigger from ht0");   // 트리거 대상 HT 지정: ht0
pid_ioctl($pid1, "set count 10 10");        // 카운트 값 설정 10, 10
pid_ioctl($pid1, "set repc 1");             // 출력 횟수 설정: 1

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

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

위 예제에서 보는 바와 같이, 출력을 동기화시킬 HT는 트리거의 대상이 되는 HT0보다 반드시 먼저 가동되어야 합니다.

출력 파형은 다음과 같습니다.


HT 0 시작
HT 0
10 μs
10 μs
HT 1
HT 1 시작
HT 1 트리거

캡쳐모드

캡쳐모드는 특정 시점으로부터 이벤트가 발생한 시점의 카운트 값을 알아내고자 할 때 사용합니다. 캡쳐모드에서 사용 가능한 명령어는 다음과 같습니다.

명령어	하위 명령어				설명
set	mode	capture	rise		캡쳐모드 설정: 상승 에지
			fall		캡쳐모드 설정: 하강 에지
			toggle		캡쳐모드 설정: 상승 또는 하강 에지
	div	ms			단위 설정: 밀리 초
		us			단위 설정: 마이크로 초
	repc	[N]			캡쳐 횟수 설정
	trigger	from	ht0		트리거 대상 설정: ht0만 가능
			php		트리거 대상 설정: 없음
			pin	rise	트리거 설정: 상승에지 핀 이벤트
				fall	트리거 설정: 하강에지 핀 이벤트
				toggle	트리거 설정: 상승/하강에지 핀 이벤트
reset	-				초기화
get	count	[N]			카운트 값 읽기
	state				상태 읽기
	repc				남은 캡쳐 횟수 읽기
start	-				시작
stop	-				정지

반복횟수 설정

캡쳐모드에서 반복횟수는 이벤트를 캡쳐할 횟수를 의미합니다. 설정 가능한 N의 범위는 0 ~ 64 입니다. 기본 값은 0이며 0은 최대 반복횟수인 64를 의미합니다.

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

트리거 설정

캡쳐모드에서는 트리거 대상으로 HT0뿐만 아니라 핀 이벤트도 설정할 수 있습니다. 캡쳐모드에서의 트리거 설정 방법은 다음과 같습니다.

구분	문법
ht0	pid_ioctl($pid, "set trigger from ht0");
핀 이벤트	pid_ioctl($pid, "set trigger from pin"); pid_ioctl($pid, "set trigger from pin rise"); pid_ioctl($pid, "set trigger from pin fall"); pid_ioctl($pid, "set trigger from pin toggle");
php	pid_ioctl($pid, "set trigger from php");

HT의 입력 핀 이벤트는 상승에지, 하강에지 및 토글(상승 또는 하강에지) 중에서 선택이 가능합니다. 핀 이벤트 트리거에서 이벤트 종류를 지정하지 않으면 기본 값으로 토글(상승 또는 하강에지)이 사용됩니다. HT는 트리거 설정 기본 값으로 트리거 대상을 지정하지 않습니다. "set trigger from php"명령을 사용하면 트리거 대상을 지정하지 않을 수 있습니다. 캡쳐모드에서 트리거 대상을 지정하지 않으면 해당 HT의 가동 시점에 캡쳐가 시작됩니다.

※ HT2는 핀 이벤트 트리거를 지원하지 않으니 유의하시기 바랍니다.

카운트 값 읽기

"get count" 명령어는 캡쳐된 HT의 카운트 값을 읽는 명령어 입니다. 뒤에 옵션으로 몇 번째 카운트 값을 읽을지를 지정합니다. 설정 방법은 다음과 같습니다.

구분	문법
N 번째 카운트 값	pid_ioctl($pid, "get count [N]");

카운트 값의 옵션은 0부터 시작합니다. 옵션을 생략하면 기본 값으로 0이 설정됩니다. 옵션 값은 최대 값이 64입니다. 캡쳐모드에서 누적 된 카운트 값은 32764를 넘을 수 없습니다. 만약 누적 카운트 값이 32764를 넘으면 HT는 즉시 캡쳐를 멈춥니다.

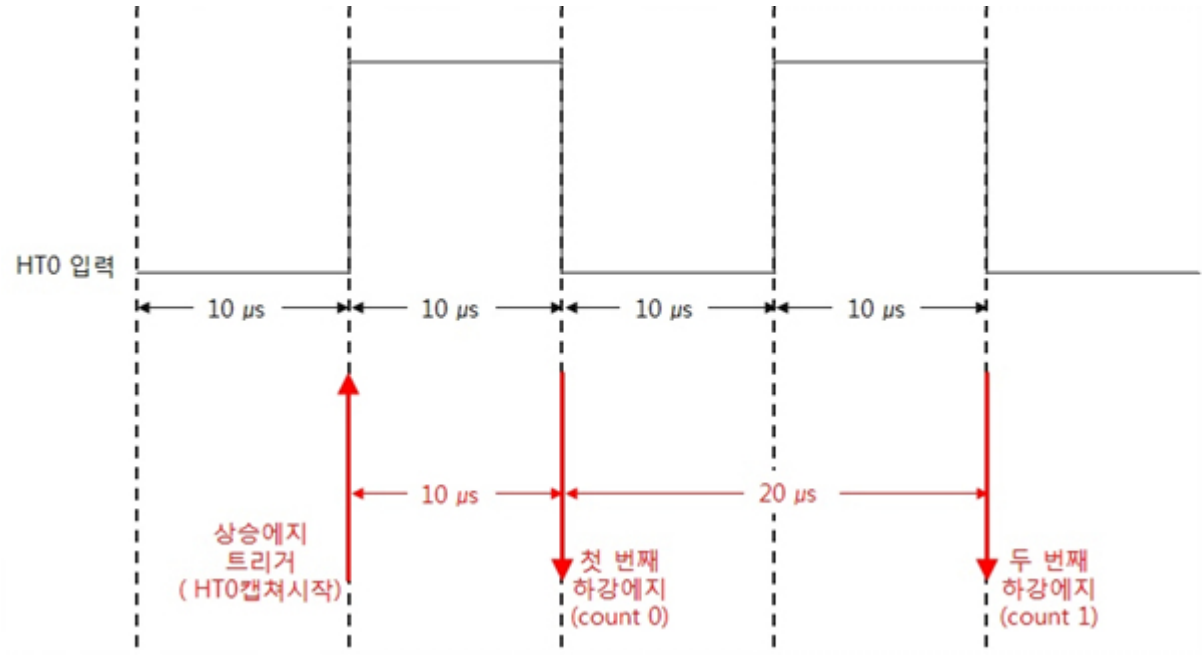
캡쳐모드와 트리거

캡쳐모드에서도 트리거 대상을 설정하여 트리거 시점을 결정합니다. 트리거 대상은 HT0뿐만 아니라 해당 HT의 핀 입력을 지정할 수 있습니다. 트리거 시점은 HT의 내부 캡쳐 카운터의 동작 시점을 의미합니다. 따라서 타이머가 동작 중이라면 트리거 시점 이전에도 캡쳐는 수행되며 이때 카운트 값들은 모두 0이 됩니다.

다음 예제는 HT입력 핀의 상승에지 이벤트에서 트리거하여 2개의 하강에지 이벤트의 카운트 값을 캡쳐하는 예제입니다.

```
<?php
$pid = pid_open("/mmap/ht0");                  // 0번 HT 열기
pid_ioctl($pid, "set div us");                 // 단위 설정: 마이크로 초
pid_ioctl($pid, "set mode capture fall");      // 캡쳐모드 설정: 하강에지
pid_ioctl($pid, "set trigger from pin rise"); // 핀 입력 트리거 설정: 상승에지
pid_ioctl($pid, "set repc 2");                 // 반복횟수 설정: 2
pid_ioctl($pid, "start");                      // 0번 HT 시작
while(pid_ioctl($pid, "get state"))
    ;
for($i = 0; $i < 2; $i++)
    echo "[$i]", pid_ioctl($pid, "get count $i"), "₩r₩n";  // 카운트 값 읽기
pid_close($pid);
?>
```

위 예제를 실행하는 동안 HT0의 입력으로 주기가 20㎲인 2개의 정사각형 펄스가 입력되었다고 가정했을 때 카운트 0과 1의 값은 다음과 같이 측정됩니다.



따라서 출력 결과는 다음과 같습니다.

```
[0]10
```

```
[1]20
```

ADC 사용 절차

일반적인 ADC 사용 절차는 다음과 같습니다.

ADC 열기

ADC를 열기 위해서는 pid_open함수를 사용합니다.

```
$pid = pid_open("/mmap/adc0");        // ADC 열기
```

※ 제품 별 ADC에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

ADC 채널 설정

ADC는 사용하기 전에 pid_ioctl함수를 이용해 채널을 설정해야 합니다. 만약 채널을 설정하지 않으면 ADC 디바이스 번호와 같은 채널 번호가 설정됩니다. 예를 들어, 0번 ADC를 채널 설정 없이 사용할 때 채널 0번이 기본 값으로 사용됩니다. ADC의 채널을 바꿔가면서 순차적으로 모든 ADC 값을 읽을 수 있습니다. 채널을 설정하거나 변경하기 위해서는 다음 명령어가 사용됩니다.

```
pid_ioctl($pid, "set ch N"); // 채널 설정
```

현재 사용중인 채널은 다음 명령어로 확인할 수 있습니다.

```
<?php
pid_ioctl($pid, "get ch");        // 채널 확인
?>
```

N은 ADC 채널 번호를 의미합니다.

ADC 채널 설정 및 확인 예

```
<?php
$pid = pid_open("/mmap/adc0");    // 0번 ADC 열기
pid_ioctl($pid, "set ch 1");      // 1번 채널로 설정
pid_ioctl($pid, "set ch 2");      // 2번 채널로 변경
echo pid_ioctl($pid, "get ch");  // 채널 확인 후 출력(출력 결과: 2)
pid_close($pid);                  // ADC 닫기
?>
```

ADC 값 확인

ADC의 값을 읽을 때에는 pid_read함수를 사용합니다.

```
<?php
pid_read($pid, $value);
?>
```

$value는 읽은 값을 저장할 정수형 변수 입니다.

기준전압은 최대 3.3V이며 기본적으로 최대 값이 사용됩니다. 만약 더 낮은 기준전압을 사용하고자 하는 경우에는 기준전압 입력 핀(AREF)에 해당 전압을 입력할 수 있습니다.

ADC 입력은 0V부터 기준전압 사이의 값이 되어야 합니다. ADC 값은 아날로그 입력을 4096단계의 디지털 값 중 하나로 판정한 결과가 됩니다.

ADC 값 읽기 예

다음 예제는 0번 ADC포트로의 아날로그 입력을 출력합니다.

```
<?php
$adc_value = 0;
$pid = pid_open("/mmap/adc0");          // 0번 ADC 열기
pid_ioctl($pid, "set ch 0");            // 0번 채널로 설정
pid_read($pid, $adc_value);             // ADC 값 읽기
echo "adc value: $adc_value₩r₩n";       // ADC 값 출력
$voltage = $adc_value * 3.3 / 4095.0;
echo "voltage : $voltage[V]₩r₩n";       // Voltage 값 출력
pid_close($pid);                        // ADC 닫기
?>
```

SPI 개요

PHPoC는 동기식 직렬 통신방식인 SPI 인터페이스를 제공합니다.

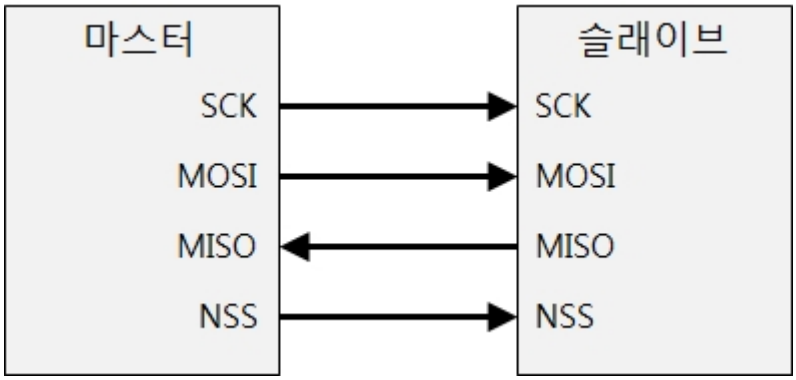
SPI 연결

SPI는 마스터와 슬레이브로 구성되며 총 4개의 연결선이 사용됩니다.

SPI 신호선

구분	의미	설명
SCK	Serial Clock	동기화를 위한 클록
MOSI	Master Output, Slave Input	마스터의 출력
MISO	Master Input, Slave Output	슬레이브의 출력
SS	Slave Select	슬레이브 선택

SPI 연결



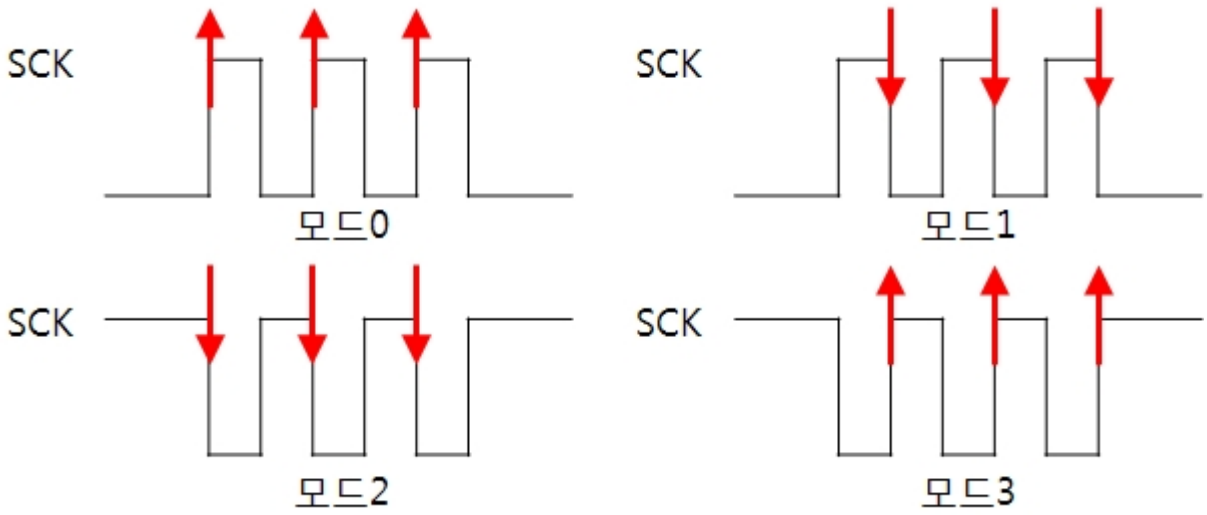
슬레이브 선택

SPI 신호선 중 SS를 제외한 나머지 세 가닥은 모든 슬레이브에 공통적으로 연결되지만, SS는 각각의 슬레이브로 연결 됩니다. 따라서 3대의 슬레이브와 통신하기 위해서는 마스터가 3개 이상의 SS포트를 가지고 있어야 합니다. 마스터가 특정 슬레이브와 데이터 통신을 하기 위해서는 해당 슬레이브가 연결 된 SS포트에는 LOW를, 나머지 SS포트들에는 HIGH를 출력합니다. 즉, 마스터는 한 번에 하나의 슬레이브와만 통신할 수 있습니다. 특정 슬레이브와 통신이 끝나면 해당 SS핀을 다시 HIGH상태로 변경합니다.

데이터 통신

SPI 모드

SPI는 샘플링 방식에 따라 모드 0 ~ 3까지 4가지 모드를 정의합니다.

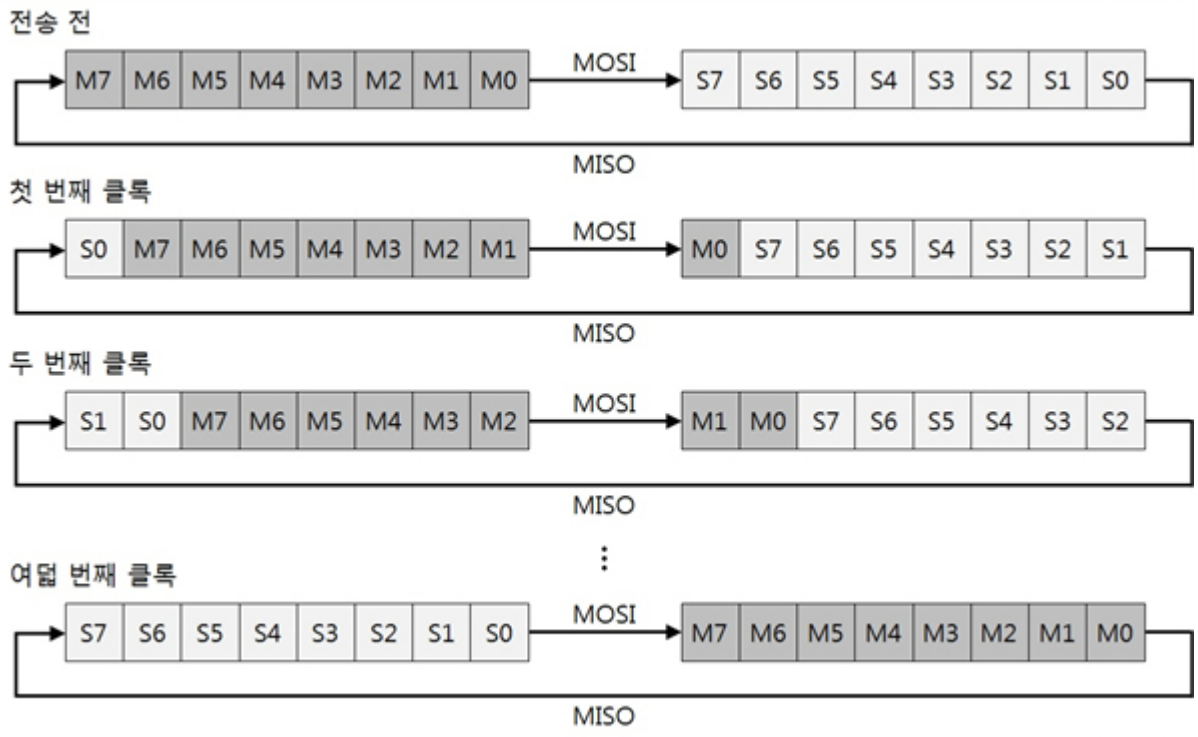


데이터 전송 순서

SPI는 데이터 통신에 앞서 마스터와 슬레이브 사이에 데이터 전송 순서를 맞춰 주어야 합니다. LSB부터 전송하는 방법과 MSB부터 전송하는 방법이 있습니다.

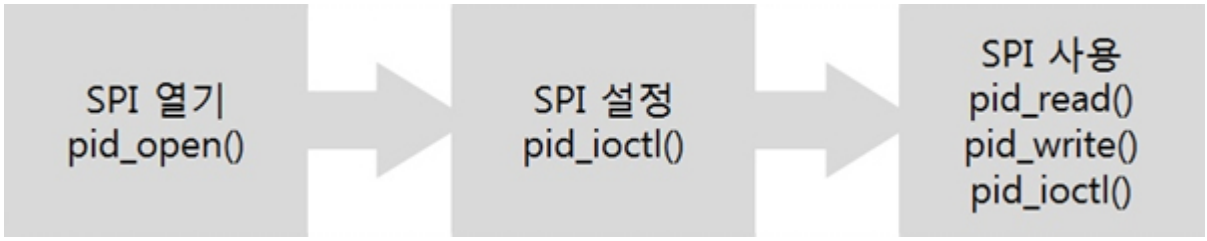
데이터 전송 흐름

SPI의 마스터와 슬레이브의 데이터 버퍼는 원형 큐의 형태입니다. 따라서 데이터 통신은 송신과 수신이 항상 동시에 이루어집니다. 다음 그림은 LSB가 먼저 전송되는 경우에 마스터와 슬레이브 사이의 데이터 송/수신을 개념적으로 나타낸 것입니다.



SPI 사용 절차

일반적인 SPI 사용 절차는 다음과 같습니다.



SPI 열기

SPI를 열기 위해서는 pid_open함수를 사용합니다.

```
<?php
$pid = pid_open("/mmap/spi0");        // SPI 열기
?>
```

※ 제품 별 제공되는 SPI에 대한 정보는 부록을 참조하시기 바랍니다.

SPI 설정 및 사용

SPI를 설정하거나 사용하기 위해서는 pid_ioctl함수를 사용해야 합니다. SPI가 지원하는 pid_ioctl함수의 명령어 목록은 다음과 같습니다.

명령어	하위 명령어				설명
set	lsb	0			데이터 전송 순서 설정: MSB 먼저 전송
		1			데이터 전송 순서 설정: LSB 먼저 전송
	data	8			데이터 전송 단위 설정: 8비트 단위 전송
		16			데이터 전송 단위 설정: 16비트 단위 전송
	div	[N]			분주비 설정, 2/4/8/16/32/64/128/256
	mode	[M]			모드 설정: 0/1/2/3
	nss	id	[N]		슬레이브 선택, 0 ~ 7
		dev	uio0	#pin	슬레이브 선택 핀(SS) 설정
get	rxlen				수신버퍼 바이트 수 확인
	txlen				송신버퍼 바이트 수 확인
	txfree				송신버퍼 여유 공간 확인
req	start				데이터 전송 요청
	reset				통신 상태 리셋

SPI 설정

SPI는 "set"명령을 이용하여 통신모드, 데이터 전송 순서, 데이터 전송 단위 및 통신 속도를 설정합니다.

데이터 전송 순서 설정

PHPoC의 SPI 데이터 전송 순서는 0또는 1 중 하나를 선택할 수 있으며 데이터 전송 순서를 설정하지 않았을 때 사용하는 기본 값은 0입니다. 0은 MSB부터 먼저 전송하는 것을 의미합니다. 설정 방법은 다음과 같습니다.

구분	문법
MSB > LSB	pid_ioctl($pid, "set lsb 0");
LSB > MSB	pid_ioctl($pid, "set lsb 1");

데이터 전송 단위 설정

PHPoC의 SPI 데이터 전송 단위는 8비트 또는 16비트 중 하나를 선택할 수 있으며 기본 값은 8입니다. 설정 방법은 다음과 같습니다.

구분	문법
8비트	pid_ioctl($pid, "set data 8");
16비트	pid_ioctl($pid, "set data 16");

데이터 통신 속도 설정

데이터 통신 속도 설정 값은 PHPoC의 기본 클록의 분주비가 되며 2 / 4 / 8 / 16 / 32 / 64 / 128 / 256 중 하나를 선택할 수 있습니다. 기본 값은 256이며 설정 방법은 다음과 같습니다.

구분	문법
N분의 1	pid_ioctl($pid, "set div N");

※ P4S-341/P4S-342는 기본 클록이 42MHz이므로 분주비가 256일 때 통신속도는 약 164Kbps 입니다.

SPI 모드

SPI모드는 0부터 3까지 중 하나를 선택하여 설정합니다. 기본 값은 3이며 설정 방법은 다음과 같습니다.

구분	문법
모드 0	pid_ioctl($pid, "set mode 0");
모드 1	pid_ioctl($pid, "set mode 1");
모드 2	pid_ioctl($pid, "set mode 2");
모드 3	pid_ioctl($pid, "set mode 3");

슬레이브 선택 및 SS 핀 설정

PHPoC가 둘 이상의 슬레이브와 통신할 때 "set nss id"명령을 이용해 슬레이브를 선택할 수 있습니다. PHPoC는 최대 8개의 SPI슬레이브와 통신할 수 있습니다. 따라서 아이디는 0 ~ 7사이에서 선택 가능하며 기본값은 0입니다.

구분	문법
N번째 슬레이브 아이디 선택	pid_ioctl($pid, "set nss N");

슬레이브 아이디 1 ~ 7을 위한 슬레이브 선택 핀(SS)은 "set nss dev"명령을 이용해 설정할 수 있습니다. 명령어 뒤에는 사용할 uio디바이스 이름과 핀번호가 지정되어야 합니다.

- SS핀 지정 예

```
pid_ioctl($pid, "set nss id 1");
pid_ioctl($pid, "set nss dev uio0 4");
pid_ioctl($pid, "set nss id 2");
pid_ioctl($pid, "set nss dev uio0 5");
...
```

※참고: 슬레이브 아이디 0의 슬레이브 선택 핀(SS)은 uio0의 0번핀이 사용되며 다른 핀으로 변경할 수 없습니다.

SPI 상태 읽기

SPI는 "get"명령을 이용하여 각종 상태정보를 확인 합니다.

송/수신 버퍼 바이트 수 읽기

SPI에서는 송신 또는 수신버퍼의 바이트 수를 다음과 같이 확인할 수 있습니다.

구분	문법
송신버퍼 바이트 수	pid_ioctl($pid, "get txlen");
수신버퍼 바이트 수	pid_ioctl($pid, "get rxlen");

송신버퍼 여유공간 확인

SPI 송신버퍼의 여유공간은 다음과 같이 확인할 수 있습니다. 반환 값의 단위는 바이트 입니다.

구분	문법
송신버퍼 여유 공간	pid_ioctl($pid, "get txfree");

SPI 사용

데이터 전송 요청

SPI 데이터를 전송하도록 요청하는 명령어 입니다. 이 명령을 수행하기 전에 반드시 pid_write함수를 이용하여 버퍼에 송신할 데이터를 넣어줘야 합니다. 또한 데이터 전송이 완료 된 후, 전송한 데이터 크기만큼 pid_read함수를 이용하여 읽어야 합니다. 사용법은 다음과 같습니다.

구분	문법
데이터 전송 요청	pid_ioctl($pid, "req start");

버스 초기화 요청

SPI 통신이 원활하지 않을 때 이 명령으로 SPI 버스를 초기화 시킬 수 있습니다.

구분	문법
버스 초기화 요청	pid_ioctl($pid, "req reset");

SPI 사용 예

슬레이브로 데이터 송신

다음은 PHPoC가 SPI 슬레이브로 데이터를 송신하는 일반적인 예 입니다.

데이터 송신 예

```
<?php
$wbuf = 0xA2;                       // 송신할 데이터
$rbuf = "";

$pid = pid_open("/mmap/spi0");      // SPI 열기
pid_ioctl($pid, "set mode 3");      // 모드 설정: 3
pid_ioctl($pid, "set lsb 0");       // 데이터 전송 순서 설정: MSB > LSB
pid_write($pid, $wbuf, 1);          // 버퍼에 1바이트 넣기: 0xA2
pid_ioctl($pid, "req start");       // 전송 요청
while(pid_ioctl($pid, "get txlen"))  // 송신 데이터 확인
    ;
pid_read($pid, $rbuf, 1);           // 버퍼의 1바이트 데이터 읽기
pid_close($pid);
?>
```

위 예에서 마지막 줄에서 pid_read함수로 1바이트를 읽은 이유는 SPI의 데이터 송신과 수신이 항상 동시에 이루어지는 특징 때문입니다.

슬레이브로부터 데이터 수신

다음은 PHPoC가 SPI 슬레이브로부터 데이터를 수신하는 일반적인 예 입니다.

데이터 수신 예

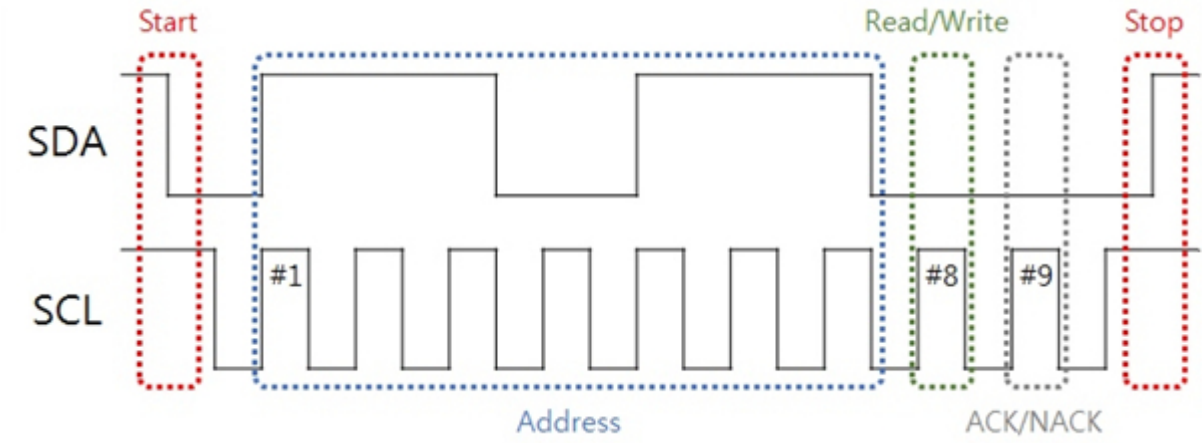
```
<?php
$wbuf = 0x00;                       // 송신 할 데이터
$rbuf = "";

$pid = pid_open("/mmap/spi0");      // SPI 열기
pid_ioctl($pid, "set mode 3");      // 모드 설정: 3
pid_ioctl($pid, "set lsb 0");       // 데이터 전송 순서 설정: MSB > LSB
pid_write($pid, $wbuf, 1);          // 버퍼에 1바이트 넣기: 0x00
pid_ioctl($pid, "req start");       // 전송 요청
while(pid_ioctl($pid, "get txlen"))  // 송신 데이터 확인
    ;
pid_read($pid, $rbuf, 1);           // 버퍼의 1바이트 데이터 읽기
pid_close($pid);
?>
```

I2C 개요

PHPoC는 2선식 직렬 버스 통신방식인 I2C를 제공합니다.

I2C 데이터 구성

I2C 데이터는 8비트 단위로 전송됩니다. 다음은 I2C 데이터의 구성을 나타낸 예 입니다.



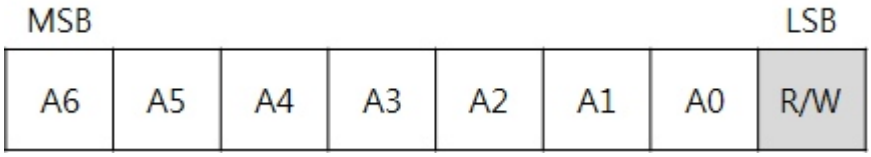
데이터 시작과 종료

I2C 데이터의 시작과 종료 조건은 다음과 같습니다.

구분	SCL	SDA
시작조건	HIGH	HIGH > LOW
종료조건	HIGH	LOW > HIGH

슬레이브 주소지정방식

PHPoC의 I2C는 7비트 주소지정방식을 사용합니다. 주소 데이터의 LSB는 읽기 또는 쓰기를 나타내는 비트 입니다.



읽기/쓰기

I2C 통신은 마스터가 읽기 또는 쓰기 프레임을 전송하면서 시작됩니다.

구분	SCL	SDA
읽기	8번째 HIGH 구간	HIGH
쓰기	8번째 HIGH 구간	LOW

응답 확인

I2C 마스터와 슬레이브는 상대방으로부터 8비트 데이터를 정상적으로 받으면 응답 확인(ACK)을 보냅니다. 정상적인 응답 확인은 SCL의 9번째 HIGH구간에서 LOW를 출력함으로써 이루어집니다. 해당 구간이

HIGH라는 것은 상대방이 데이터를 수신하지 못했다는 것을 의미합니다.

구분	SCL	SDA
응답 확인(ACK)	9번째 HIGH 구간	LOW
응답 미확인(NACK)	9번째 HIGH 구간	HIGH

I2C 데이터 통신 시나리오

다음은 4개의 I2C 데이터 통신 시나리오 입니다. 음영 표시가 되지 않는 부분은 마스터가 전송한 데이터 이고, 음영 표시가 된 부분은 슬레이브가 전송한 데이터 입니다.

쓰기 데이터 전송 성공

시작	디바이스 주소	R/W	ACK	쓰기 데이터	ACK	종료
	1 1 1 0 1 1 1	0	0	0 0 0 1 1 1 1 0	0	

쓰기 데이터 전송 실패

시작	디바이스 주소	R/W	ACK	종료
	1 1 1 0 1 1 1	0	1	

읽기 데이터 전송 성공

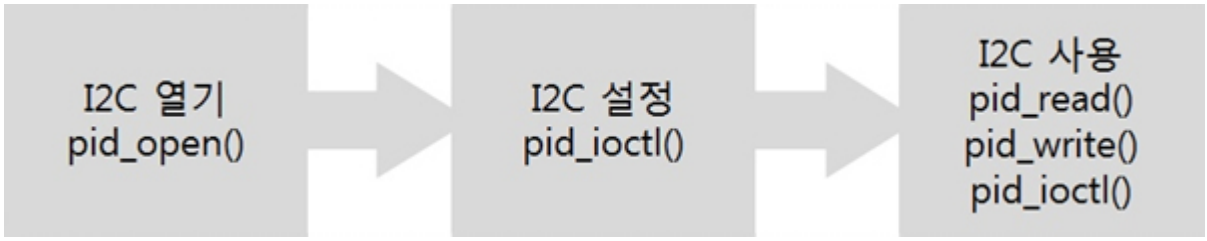
시작	디바이스 주소	R/W	ACK	응답 데이터	ACK	종료
	1 1 1 0 1 1 1	1	0	0 1 1 0 0 1 1 0	0	

읽기 데이터 전송 실패

시작	디바이스 주소	R/W	ACK	종료
	1 1 1 0 1 1 1	1	1	

I2C 사용 절차

일반적인 I2C 사용 절차는 다음과 같습니다.



I2C 열기

I2C를 열기 위해서는 pid_open함수를 사용합니다.

```
<?php
$pid = pid_open("/mmap/i2c0");       // I2C 열기
?>
```

※ 제품 별 제공되는 I2C에 대한 정보는 부록을 참조하시기 바랍니다.

I2C 설정 및 사용

I2C를 설정하거나 사용하기 위해서는 pid_ioctl함수를 사용해야 합니다. I2C가 지원하는 pid_ioctl함수의 명령어 목록은 다음과 같습니다.

명령어	하위 명령어		설명
set	mode	sm	모드 설정: 표준 모드 - 100 Kbps
		fm	모드 설정: 고속 모드 - 400 Kbps
	daddr	[A]	주소 설정: 로컬 슬레이브 디바이스 주소
	saddr	[A]	주소 설정: 상대 슬레이브 디바이스 주소
get	rxlen		수신버퍼 바이트 수 확인
	txlen		송신버퍼 바이트 수 확인
	txfree		송신버퍼 여유 공간 확인
	error	nack	마지막 트랜잭션의 NACK 횟수 확인
		bus	마지막 트랜잭션의 BUS 오류 횟수 확인
	state		0 - 유휴상태, 그 외 - I2C 내부 동작 중
req	read	[N]	읽기 요청
	write	-	쓰기 요청
		wait	쓰기 요청 및 종료 대기 (후에 "req stop" 필요)
	stop		쓰기 종료 (전에 "req wait" 필요)
	reset		통신 상태 리셋

I2C 설정

I2C는 "set"명령을 이용하여 통신모드와 슬레이브 디바이스 주소를 설정합니다.

통신속도 설정

PHPoC의 I2C는 표준모드와 고속모드를 지원하며 기본 값은 표준모드 입니다.

구분	문법
표준모드	pid_ioctl($pid, "set mode sm");
고속모드	pid_ioctl($pid, "set mode fm");

디바이스 주소 설정

I2C는 데이터를 전송할 슬레이브를 선택할 때 슬레이브 디바이스의 주소를 이용합니다. 슬레이브 디바이스 주소는 다음과 같이 설정할 수 있습니다.

구분	문법
슬레이브 디바이스 주소	pid_ioctl($pid, "set saddr [A]");
로컬 디바이스 주소	pid_ioctl($pid, "set daddr [A]");

디바이스 주소 [A]는 16진수 2자리 형태로 입력해야 합니다. PHPoC의 I2C는 7비트 주소 지정방식을 사용하므로 LSB는 항상 0입니다. 참고로 몇몇 예약 된 주소들은 사용할 수 없습니다. 만일 이 주소들을 사용하면 PHPoC 에러가 발생합니다.

사용 불가능한 주소(2진수)								예(16진수)	비고
7	6	5	4	3	2	1	0		
X	X	X	X	X	X	X	1	E1, A3, 1B	LSB가 1
0	0	0	0	X	X	X	X	00 ~ 0F	상위 4비트가 모두 0
1	1	1	1	X	X	X	X	F0 ~ FF	상위 4비트가 모두 1

I2C 상태 읽기

I2C는 "get"명령을 이용하여 각종 상태정보를 확인 합니다.

송/수신버퍼 바이트 수 읽기

I2C에서는 송신 또는 수신버퍼의 바이트 수를 다음과 같이 확인할 수 있습니다.

구분	문법
송신버퍼 바이트 수	pid_ioctl($pid, "get txlen");
수신버퍼 바이트 수	pid_ioctl($pid, "get rxlen");

송신버퍼 여유공간 확인

I2C 송신버퍼의 여유공간은 다음과 같이 확인할 수 있습니다. 반환 값의 단위는 바이트 입니다.

구분	문법
송신버퍼 여유 공간	pid_ioctl($pid, "get txfree");

오류 횟수 읽기

I2C에서는 마지막 트랜잭션의 NACK 또는 버스 에러 횟수를 확인할 수 있습니다.

구분	문법
NACK	pid_ioctl($pid, "get error nack");
버스 에러	pid_ioctl($pid, "get error bus");

상태 읽기

I2C의 상태는 다음과 같이 확인할 수 있습니다. I2C는 유휴 상태일 때 0을 반환하고, 유휴상태가 아닌 경우에는 0이 아닌 값을 반환합니다.

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

I2C 사용

읽기 요청

I2C 마스터가 슬레이브로 읽기 데이터를 전송하도록 요청하는 명령어 입니다. 이 명령을 수행한 후 슬레이브의 응답 데이터가 정상적으로 버퍼에 수신되면 pid_read함수를 이용하여 그 내용을 읽을 수 있습니다.

구분	문법
읽기 요청	pid_ioctl($pid, "req read [N]");

위 문법에서 [N]은 읽고자 하는 데이터의 바이트 수를 의미합니다.

쓰기 요청

I2C 마스터가 슬레이브로 쓰기 데이터를 전송하도록 요청하는 명령어 입니다. 쓰기 요청에는 다음과 같이 두 가지 방법이 있습니다.

구분	문법
쓰기 요청	pid_ioctl($pid, "req write");
쓰기 요청 및 종료 대기	pid_ioctl($pid, "req write wait");
쓰기 종료 요청	pid_ioctl($pid, "req stop");

"req write"명령은 수행되면 바로 데이터를 송신합니다. 따라서 명령 수행 전에 반드시 버퍼에 송신할 데이터를 넣어줘야 합니다. 반면 "req write wait"명령은 버스 상태를 전송 시작 조건으로 만들고 "req stop"명령이 수행 될 때까지 종료 조건으로 만들지 않습니다. 따라서 데이터 "req stop"명령이 수행되기 전까지 pid_write함수를 이용해 계속해서 데이터를 전송할 수 있습니다.

초기화 요청

버스 에러 등에 의해 통신이 원활하지 않을 때 이 명령으로 I2C 통신을 초기화 시킬 수 있습니다.

구분	문법
초기화 요청	pid_ioctl($pid, "req reset");

I2C 사용 예

슬레이브로 데이터 쓰기

다음은 PHPoC가 I2C 슬레이브로 데이터를 쓰는 일반적인 예 입니다.

쓰기 요청 예

```
<?php
$wbuf = 0x7A;
$pid = pid_open("/mmap/i2c0");       // I2C 열기
pid_ioctl($pid, "set mode fm");      // 고속모드 설정
pid_ioctl($pid, "set saddr ee");     // 슬레이브 디바이스 주소 설정: 0xEE
pid_write($pid, $wbuf, 1);           // 버퍼에 1바이트 넣기: 0x7A
pid_ioctl($pid, "req write");        // 쓰기 요청
while(pid_ioctl($pid, "get txlen")) // 송신 데이터 확인
    ;
pid_close($pid);
?>
```

쓰기 요청 대기 및 수행 예

```
<?php
$pid = pid_open("/mmap/i2c0");          // I2C 열기
pid_ioctl($pid, "set mode fm");         // 고속모드 설정
pid_ioctl($pid, "set saddr ee");        // 슬레이브 디바이스 주소 설정: 0xEE
pid_ioctl($pid, "req write wait");      // 쓰기 요청 및 종료 대기
pid_write($pid, 0x7A, 1);               // 1바이트 송신: 0x7A
pid_write($pid, 0x8A, 1);               // 1바이트 송신: 0x8A
pid_write($pid, 0x9A, 1);               // 1바이트 송신: 0x9A
while(pid_ioctl($pid, "get txlen"))     // 송신 데이터 확인
    ;
pid_ioctl($pid, "req stop");            // 쓰기 종료 요청
pid_close($pid);
?>
```

슬레이브로부터 데이터 읽기

다음은 PHPoC가 I2C 슬레이브의 데이터를 읽는 일반적인 예 입니다.

읽기 요청 예

```
<?php
$rbuf = "";
$pid = pid_open("/mmap/i2c0");          // I2C 열기
```

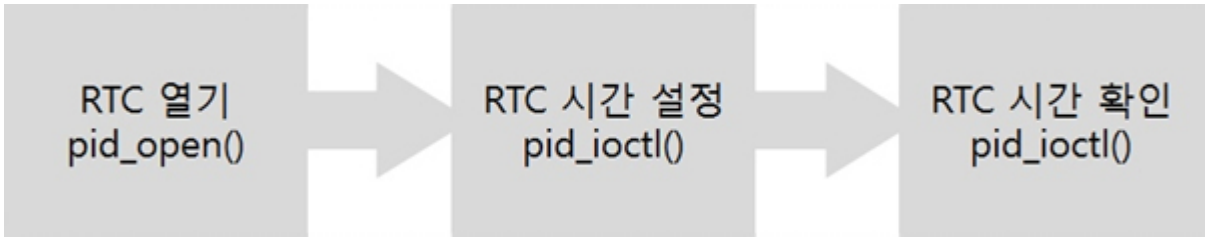
```
pid_ioctl($pid, "set mode fm");          // 고속모드 설정
pid_ioctl($pid, "set saddr ee");         // 슬레이브 디바이스 주소 설정: 0xEE
pid_ioctl($pid, "req read 2");           // 2바이트 읽기 요청
while(pid_ioctl($pid, "get rxlen") < 2) // 수신 데이터 확인
    ;
pid_read($pid, $rbuf);                   // 버퍼의 데이터 읽기
pid_close($pid);
?>
```

RTC 개요

PHPoC는 정확한 시간을 유지하기 위해 RTC(Real Time Clock)를 제공합니다. RTC의 시간은 내장 배터리에 의해 동작되기 때문에 제품의 전원이 연결되지 않더라도 계속 동작합니다.

RTC 사용 절차

일반적인 RTC 사용 절차는 다음과 같습니다.



RTC 열기

RTC를 열기 위해서는 pid_open함수를 사용합니다.

```
<?php
$pid = pid_open("/mmap/rtc0");        // RTC 열기
?>
```

※ 제품 별 RTC에 관한 자세한 내용은 부록을 참조하시기 바랍니다.

RTC 시간 설정

RTC에 현재 시간을 설정하기 위해서는 pid_ioctl함수의 "set date"명령어를 사용합니다.

```
pid_ioctl($pid, "set date TIME");
```

TIME값은 문자열 형태이며 설정 형식은 다음과 같습니다.

구분	년	월	일	시	분	초
형식	YYYY	MM	DD	hh	mm	ss
예1	2000	01	03	03	05	07
예2	2010	12	28	19	59	16

다음은 RTC에 현재 시간을 설정하는 예 입니다.

RTC 설정 예

```
<?php
$pid = pid_open("/mmap/rtc0");       // RTC 열기
$date = "20160720135607";            // 2016년 7월 20일 13시 56분 7초
pid_ioctl($pid, "set date $date");  // 시간설정
pid_close($pid);
?>
```

RTC 시간 확인

RTC로부터 시간을 읽어오기 위해서는 pid_ioctl함수를 사용합니다.

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

확인 가능한 RTC 정보

ITEM	설명	반환 값	반환 형식
date	시간(날짜 포함)	예) 20160720135607	문자열
wday	요일	0: 일, 1: 월, 2: 화, 3: 수, 4: 목, 5: 금, 6: 토	정수

RTC값 읽기 예

```
<?php
$date = "";
$wday = 0;
$pid = pid_open("/mmap/rtc0");              // RTC 열기
$date = pid_ioctl($pid, "get date");        // 시간 읽기
$wday = pid_ioctl($pid, "get wday");        // 요일 읽기
echo "date: $date₩r₩n", "wday: $wday₩r₩n"; // 시간 및 요일 출력
pid_close($pid);
?>
```

※ RTC의 시간 반환 형식은 설정 형식과 동일합니다.

"date" 함수

PHPoC는 RTC의 값을 읽는 date라는 내부함수를 제공합니다. 이 함수를 이용하면 RTC 값을 원하는 형태로 읽을 수 있습니다.

date함수를 이용한 RTC값 읽기 예

```
<?php
$date1 = date("Y-m-d H:i:s");
$date2 = date("D M j H:i:s Y");
echo "$date1₩r₩n";                 // 출력 예) 2016-07-20 13:56:07
echo "$date2₩r₩n";                 // 출력 예) Wed Jul 20 13:56:07 2016
?>
```

※ date함수에 대한 보다 자세한 내용은 PHPoC Internal Functions 매뉴얼을 참조하시기 바랍니다.

디바이스 관련 내부함수

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]);

※ 각 함수에 대한 자세한 내용은 PHPoC Internal Functions문서를 참조하시기 바랍니다.

디바이스 정보

제품 별 디바이스 개수

구분	P4S-341 / P4S-342	P4M-400
UART	2	2
NET	1	2
TCP	5	5
UDP	5	5
UIO	1 (24 CH)	2 (27 CH)
ST	8	8
HT	4	4
ADC	2 (6 CH)	2 (4 CH)
RTC	1	1
SPI	1	1
I2C	1	1

제품 별 디바이스 파일 경로

UART

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/uart0
	/mmap/uart1

NET

구분	파일 경로	비고
P4S-341 / P4M-400	/mmap/net0	유선랜
P4S-342 / P4M-400	/mmap/net1	무선랜

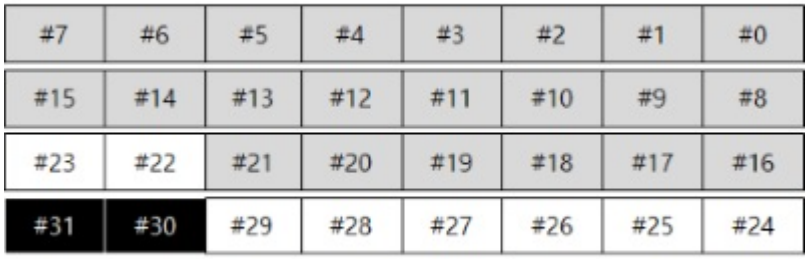
TCP

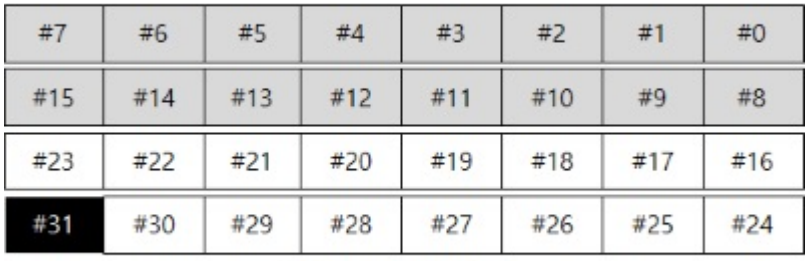
구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/tcp0
	/mmap/tcp1
	/mmap/tcp2
	/mmap/tcp3
	/mmap/tcp4

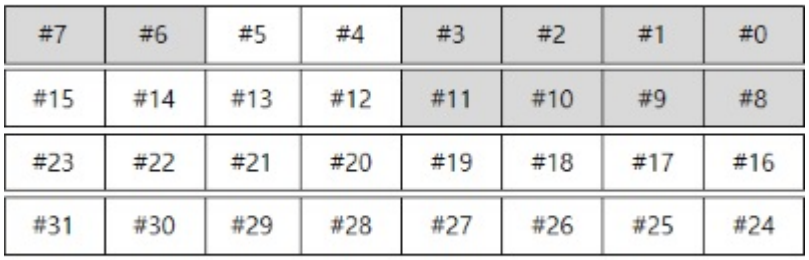
UDP

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/udp0
	/mmap/udp1
	/mmap/udp2
	/mmap/udp3
	/mmap/udp4

I/O

<table>
<tr><th>구분</th><th>파일 경로 및 맵핑 정보</th></tr>
<tr><td>P4S-341 / P4S-342</td><td>/mmap/uio0
DIO LED 할당되지 않음
 

"/mmap/uio0"</td></tr>
<tr><td>P4M-400</td><td>/mmap/uio0
DIO LED 할당되지 않음
 

"/mmap/uio0"
/mmap/uio1
DIO 할당되지 않음
 

"/mmap/uio1"</td></tr>
</table>

ST

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/st0
	/mmap/st1
	/mmap/st2
	/mmap/st3
	/mmap/st4
	/mmap/st5
	/mmap/st6
	/mmap/st7

HT

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/ht0
	/mmap/ht1
	/mmap/ht2
	/mmap/ht3

ADC

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/adc0
	/mmap/adc1

SPI

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/spi0

I2C

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/i2c0

RTC

구분	파일 경로
P4S-341 / P4S-342 / P4M-400	/mmap/rtc0

ENV 및 사용자메모리

구분		파일 경로	크기(바이트)
P4S-341 / P4S-342 / P4M-400	System ENV	/mmap/envs	1536
	User ENV	/mmap/envu	1536
	User Memory	/mmap/um0	64
		/mmap/um1	64
		/mmap/um2	64
		/mmap/um3	64
	Nonvolatile Memory	/mmap/nm0	2048

펌웨어 사양 및 제한사항

펌웨어 적용 제품

펌웨어	적용 제품
P40	P4S-341, P4S-342, P4M-400

펌웨어 사양

항목	사양	설명
ENVS	1,536	시스템 ENV 크기, byte
ENVU	1,536	사용자 ENV 크기, byte
WLAN	1	무선랜
EMAC	1	이더넷
UART	2	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
HT	4	하드웨어 타이머
ADC	2	아날로그 입력(ADC)
SPI	1	SPI
I2C	1	I2C
RTC	1	RTC

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

구분	제한사항
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

변경 내역

201021 (F/W: 2.3.1)

- SSL method 삭제: tls1_client, tls1_server
- TCP API 삭제: TELNET, SSH
- TCP client 접속관련 예제 개선
- 부록 > 변경 내역 페이지 추가