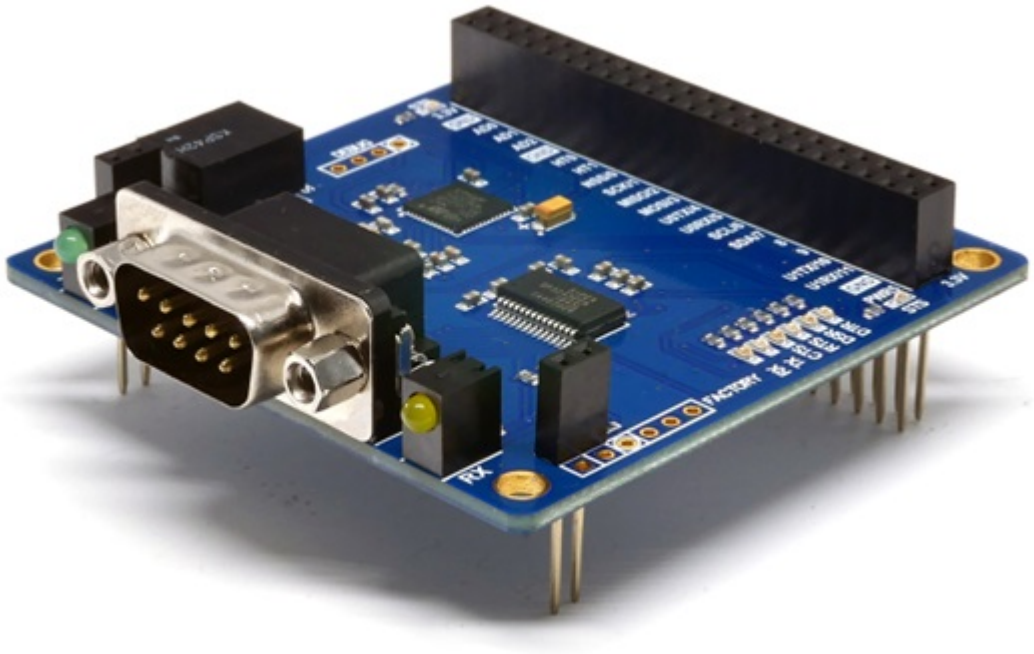
제품 소개



PES-2406

스마트 RS-232 보드 PES-2406은 PHPoC 보드형 제품 전용 스마트 확장보드입니다. 이 보드를 이용하면 PHPoC보드에 RS-232 통신기능을 쉽게 구현할 수 있습니다.

PES-2406의 주요 특징

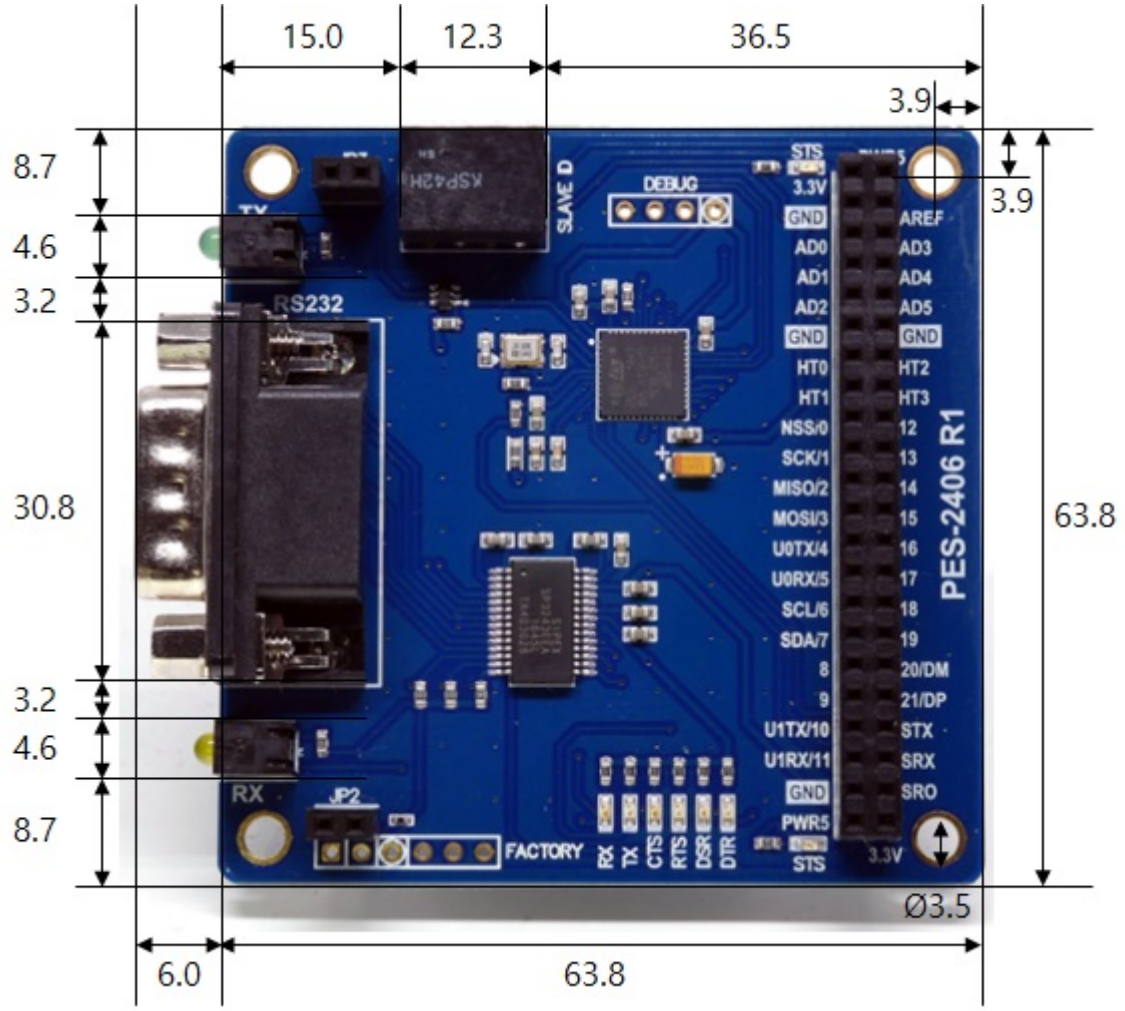
- 1 X RS-232 포트: 1200bps ~ 115200bps
- H/W 및 S/W 흐름제어 지원
- 프레임 구분자기능 지원
- 프레임 간격 설정기능 제공
- 소비 전류: 약 25[mA]

※ 주의: PES-2406을 사용하기 위해서는 펌웨어 버전 1.3.0이상의 PHPoC 보드가 필요합니다!

> 스마트 확장보드란?
>
> 스마트 확장보드는 일반 확장보드와는 달리 PHPoC보드의 디바이스 및 펌웨어와는 독립적인 자체 디바이스와 전용 펌웨어를 내장하고 있습니다. 이 보드는 PHPoC 보드와 전용 통신 포트를 이용해 마스터-슬레이브 방식으로 통신합니다. 하나의 PHPoC 보드에 여러 개의 스마트 확장보드를 연결할 수 있으며 각각의 스마트 확장보드에는 반드시 슬레이브 아이디를 설정해야 합니다.

치수

제품 본체



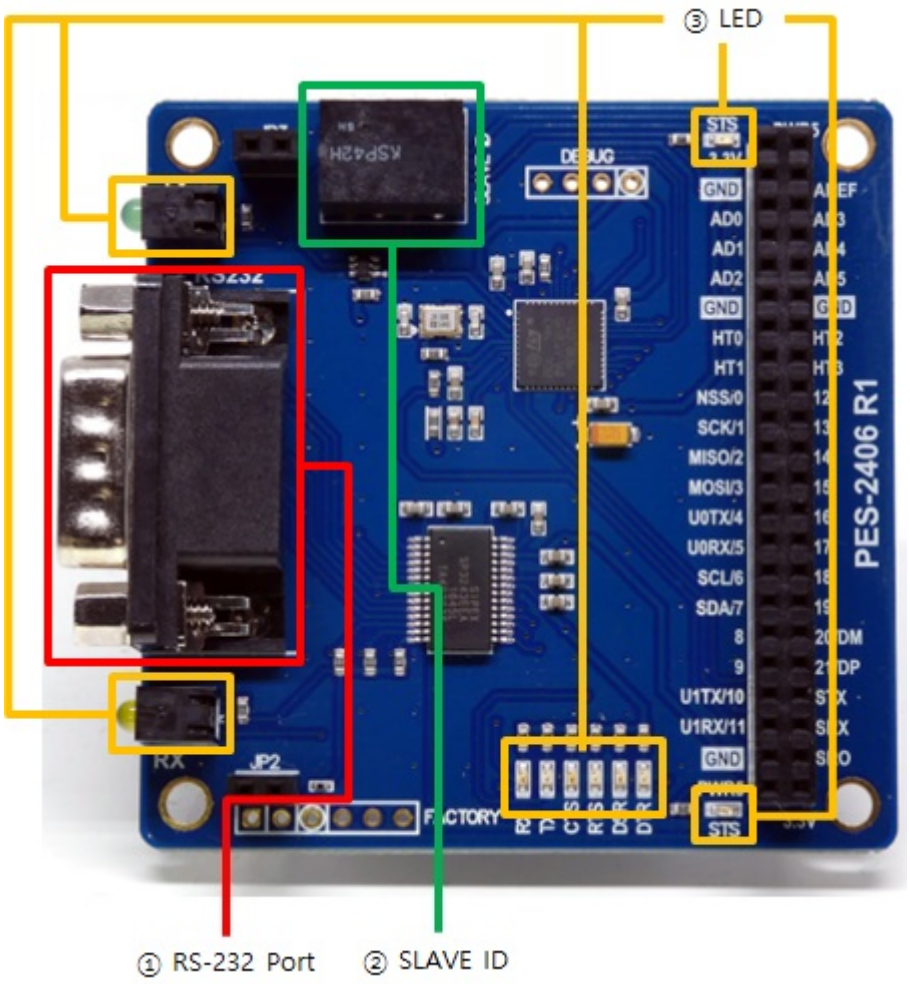
※ 치수(단위 : mm)는 제품 상태 및 재는 각도 등에 따라 약간의 오차가 있을 수 있습니다.

회로도

PES-2406의 회로도 입니다.

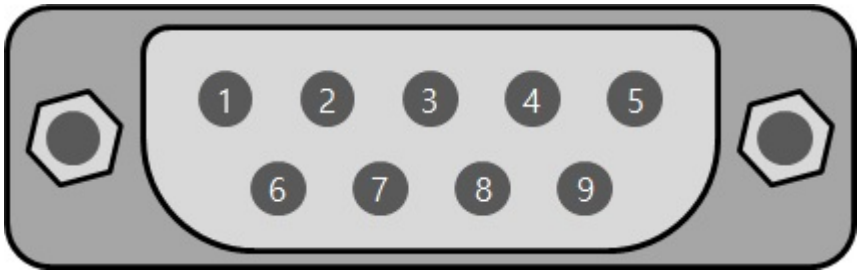
- PES-2406-R1-PO.pdf

레이아웃


③ LED
STS
DEBUG
GND
AD0
AD1
AD2
GND
HT0
HT1
NSS/0
SCK/1
MISO/2
MOSI/3
U0TX/4
U0RX/5
SCL/6
SDA/7
8
9
U1TX/10
U1RX/11
GND
PES-2406 R1
RX
JP2
FACTORY
STS
① RS-232 Port
② SLAVE ID

1. RS-232 포트

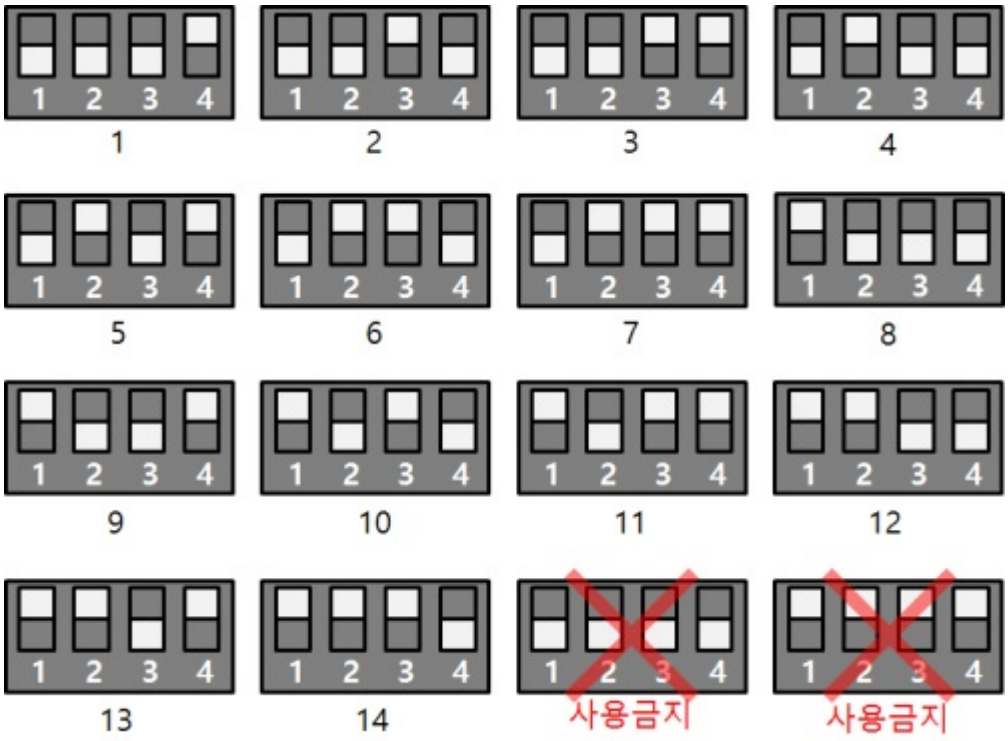
PES-2406의 RS-232 포트는 D-SUB 9핀 Male(수) 커넥터로 되어 있으며 핀 맵은 다음과 같습니다.



번호	이름	설명	신호레벨	방향	외부결선
1	DCD	Data Carrier Detect	RS-232	입력	선택
2	RXD	Receive Data	RS-232	입력	필수
3	TXD	Transmit Data	RS-232	출력	필수
4	DTR	Data Terminal Ready	RS-232	출력	선택
5	GND	Ground	Ground	-	필수
6	DSR	Data Set Ready	RS-232	입력	선택
7	RTS	Request To Send	RS-232	출력	선택
8	CTS	Clear To Send	RS-232	입력	선택
9	RI	Ring Indicator	RS-232	입력	선택

2. 슬레이브 아이디 스위치

슬레이브 아이디는 마스터인 PHPoC 보드가 스마트 확장보드 각각을 구분하는데 사용됩니다. 하나의 PHPoC 보드에 연결되는 각 스마트 확장보드는 고유한 슬레이브 아이디를 사용해야 합니다.
슬레이브 아이디는 1부터 14까지 14개 중 하나로 설정할 수 있으며 다음과 같이 4개의 딥스위치를 조정하여 설정합니다.



3. LED

PES-2406에는 총 10개의 LED가 있습니다. 이 중 2개는 보드의 상태를 나타내는 STS LED 입니다. JP1 상단의 STS LED는 3.3V에, 하단의 STS LED는 5V와 연결되어 있습니다. 나머지 8개는 시리얼통신의 각종 상태를 나타내는 LED입니다.

각 LED의 동작과 의미는 다음과 같습니다.

구분	개수	종류	색	동작 설명
STS	2	SMD	빨간색	아이디 설정이 정상일 때 > 1초마다 켜짐/꺼짐 반복 아이디 설정이 올바르지 않을 때 > 빠르게 깜박임
TX	2	DIP, SMD	초록색	시리얼포트로 데이터 송신시 깜박임
RX	2	DIP, SMD	노란색	시리얼포트로 데이터 수신시 깜박임
CTS	1	SMD	노란색	CTS신호가 ON일 때 켜짐
RTS	1	SMD	초록색	RTS신호가 ON일 때 켜짐
DSR	1	SMD	노란색	DSR신호가 ON일 때 켜짐
DTR	1	SMD	초록색	DTR신호가 ON일 때 켜짐

사용하기

PES-2406을 사용하는 방법은 다음과 같습니다.

1. PHPoC 보드에 연결

PES-2406은 단독으로 사용할 수 없습니다. 반드시 PHPoC 보드에 연결하여 사용하시기 바랍니다.

2. 소프트웨어(IDE) 설치

PHPoC 디버거는 PHPoC 제품의 설정 및 개발에 사용되는 소프트웨어 입니다. PES-2406은 PHPoC 보드형 제품을 통해 제어할 수 있으므로 이 보드를 사용하기 위해서는 PC에 PHPoC 디버거를 반드시 설치해야 합니다.

- PHPoC 디버거 다운로드 페이지
- PHPoC 디버거 매뉴얼 페이지

3. SPC라이브러리 및 예제코드 활용

SPC라이브러리는 PES-2406을 비롯한 스마트 확장보드 라이브러리 입니다. 이 라이브러리를 사용하면 비교적 간단하게 PES-2406을 사용할 수 있습니다. 라이브러리와 함수에 대한 자세한 내용은 다음 문서를 참조하시기 바랍니다.

- SPC라이브러리 매뉴얼 페이지

명령어

스마트 확장보드를 사용하기 위해서는 spc_request, spc_request_dev 및 spc_request_sys함수가 필요합니다.

```
spc_request($sid, 6, $wbuf)
spc_request($sid, 7, $rbuf)
spc_request_dev($sid, $cmd)
spc_request_sys($sid, $cmd)
```

- $sid: 보드에 설정된 슬레이브 아이디
- $wbuf: 송신 데이터가 저장된 버퍼
- $rbuf: 수신 데이터를 저장할 버퍼
- $cmd: 명령어 문자열

스마트 확장보드 공통 명령어

모든 스마트 확장보드가 공통으로 지원하는 명령어는 spc_request_sys함수를 사용하며, 명령어 목록은 다음과 같습니다.

명령어	인수	설명
get	did	디바이스 아이디 확인
get	uid	유니크 아이디 확인

PES-2406 명령어

PES-2406 전용 명령어는 spc_request 또는 spc_request_dev함수를 사용합니다. 데이터 송신 및 수신은 spc_request함수를 사용하며 설정 및 상태를 확인하기 위한 명령어들은 spc_request_dev함수를 사용합니다.

PES-2406의 명령어 목록은 다음과 같습니다.

명령어	인수1	인수2	인수3
set	uart	(parameters)	-
	modem	(signal)	-
		rts	(rts signal)
		dtr	(dtr signal)
	count	(counter)	(value)
	ifg	(bits)	-
	ifd	(del)	-
		(start_del)	(end_del)
		-	-
	txdelay	(bits)	-
	break	(time)	-

get	uart	-	-
	modem	(signal)	-
	count	(counter)	-
	rxlen	[del]	-
	txfree	-	-
	rxbuf	-	-
	txbuf	-	-
	ifg	-	-
	ifd	-	-
	txdelay	-	-

※ (): 인수 사용 필수, []: 인수 생략 가능

설정하기

PES-2406을 설정하기 위한 명령어는 set입니다.

통신 파라미터 설정

RS-232의 통신 파라미터를 설정하는 명령어는 uart입니다.

```
"set uart (parameter)"
```

parameter에 다음과 같은 형태의 문자열을 입력합니다.

```
"(baudrate)[parity[data bit[stop bit[flow control]]]]"
```

※ (): 사용 필수, []: 생략 가능

파라미터	설정 범위	설명	기본 값
baudrate	1200 ~ 115200	통신 속도(bps)	115200
parity	N, E, O, M 또는 S	패리티 비트 (N: 없음, E: 짝수, O: 홀수, M: Mark, S: Space)	N
data bit	8 또는 7	데이터 비트	8
stop bit	1 또는 2	정지 비트	1
flow control	N, H 또는 S	흐름제어 (N: 없음, H: RTS/CTS, S: Xon/Xoff)	N

- 통신 파라미터 설정 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200");
echo spc_request_dev($sid, "get uart"), "₩r₩n";   // output: 115200N81N

spc_request_dev($sid, "set uart 115200N81");
echo spc_request_dev($sid, "get uart"), "₩r₩n";   // output: 115200N81N

spc_request_dev($sid, "set uart 9600E72H");
echo spc_request_dev($sid, "get uart");           // output: 9600E72H

?>
```

※ 주의: 흐름제어(H또는 S)와 프레임 간격 설정("set ifg")은 동시에 사용할 수 없습니다.

모뎀 회선 신호 설정

모뎀 회선 신호를 설정하는 명령어는 modem입니다. 이 명령으로 제어할 수 있는 모뎀 회선 신호는 RTS와 DTR이며 두 신호를 동시에 설정하거나 각각 설정할 수 있습니다.

동시에 설정

```
"set modem (signal)"
```

signal에 2진수 2자리를 입력합니다.
첫 번째 자리는 RTS신호의 값을, 두 번째 자리는 DTR신호의 값을 의미합니다. 값 0은 active상태를 나타내고 값 1은 inactive상태를 나타냅니다.

설정 값	RTS상태	DTR상태
00	active	active
01	active	inactive
10	inactive	active
11	inactive	inactive

- 동시에 설정 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set modem 11");                  // RTS & DTR: active
echo spc_request_dev($sid, "get modem"), "₩r₩n";       // output(e.g.): 111111
sleep(1);

spc_request_dev($sid, "set modem 00");                  // RTS & DTR: inactive
echo spc_request_dev($sid, "get modem"), "₩r₩n";       // output(e.g.): 110101
sleep(1);
?>
```

각각 설정

```
"set modem rts (signal)"
"set modem dtr (signal)"
```

signal에 2진수 1자리를 입력합니다.
값 0은 active상태를 나타내고 값 1은 inactive상태를 나타냅니다.

- 각각 설정 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set modem rts 1");              // RTS: active
echo spc_request_dev($sid, "get modem rts"), "₩r₩n";  // output(e.g.): 1
sleep(1);

spc_request_dev($sid, "set modem dtr 1");              // DTR: active
echo spc_request_dev($sid, "get modem dtr");           // output(e.g.): 1
?>
```

※ 주의: 모뎀 회선 신호 설정("set modem")과 하드웨어 흐름제어(RTS/CTS)는 동시에 사용할 수 없습니다.

내부 카운터 값 설정

내부 카운터 값을 설정하는 명령어는 count입니다.

> "set count (counter) (value)"

counter에 값을 설정할 내부 카운터 이름을, value에 설정할 카운터 값을 각각 입력합니다.

내부 카운터	설명
rx	수신 바이트 카운터
tx	송신 바이트 카운터
rf	수신 프레임 카운터
tf	송신 프레임 카운터
pe	패리티 에러 카운터
fe	프레이밍 에러 카운터
oe	오버런 에러 카운터
be	브레이크 에러 카운터
rbo	수신 버퍼 오버플로우 카운터
tbo	송신 버퍼 오버플로우 카운터
rfo	수신 프레임 포인터 오버플로우 카운터
tfo	수신 프레임 포인터 오버플로우 카운터

- 내부 카운터 값 설정 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set count rx 0");
spc_request_dev($sid, "set count tx 0");
spc_request_dev($sid, "set count rf 0");
spc_request_dev($sid, "set count tf 0");
spc_request_dev($sid, "set count pe 0");
spc_request_dev($sid, "set count fe 0");
spc_request_dev($sid, "set count oe 0");
spc_request_dev($sid, "set count be 0");
spc_request_dev($sid, "set count rbo 0");
spc_request_dev($sid, "set count tbo 0");
spc_request_dev($sid, "set count rfo 0");
spc_request_dev($sid, "set count tfo 0");
?>
```

프레임 간격 설정

프레임 간격을 설정하는 명령어는 ifg입니다.

```
"set ifg (bits)"
```

bits에 프레임 간격을 설정합니다.

프레임 간격 설정은 시간을 이용해 프레임을 구분하는 방식입니다. 설정 단위는 비트(bits)이며 최소 0에서 최대 30000까지 설정할 수 있습니다. 예를 들어 9600bps에서 프레임 간격을 10으로 설정한다면 실제 설정시간은 약 0.001초(=10/9600)가 됩니다. 단위가 비트이므로 같은 값으로 설정하더라도 통신속도가 다르면 설정시간은 달라집니다.

데이터 송신시

송신되는 프레임들은 설정한 프레임 간격으로 전송됩니다.

- 프레임 간격 설정 및 데이터 송신 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");
// set frame to frame interval : 100 milliseconds
spc_request_dev($sid, "set ifg 11520");

spc_request($sid, 7, "This is the first frame.₩r₩n");
spc_request($sid, 7, "This is the second frame.₩r₩nIt will be transmitted 100 milliseconds later right after the first packet has been transmitted. ₩r₩n");
?>
```

데이터 수신시

데이터를 수신하다가 설정한 시간동안 새로운 데이터가 없으면 그 시점까지 수신한 데이터를 하나의 프레임으로 인식합니다. 이 때 "get rxlen" 명령으로 프레임 길이를 확인하여 프레임단위로 데이터를 수신할 수 있습니다.

- 프레임 간격 설정 및 데이터 수신 예

```
<?php
include "/lib/sd_spc.php";
```

```
$rbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");
// set frame to frame interval : 100 milliseconds
spc_request_dev($sid, "set ifg 11520");

while(1)
{
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        $rbuf = spc_request($sid, 6, "$rlen");
        echo "frame length = $rlen₩r₩n";
        hexdump($rbuf);
    }
}
?>
```

※ 주의: 프레임 간격 설정("set ifg")과 흐름제어(H 또는 S)는 동시에 사용할 수 없습니다.
※ 주의: 프레임 간격 설정("set ifg")과 프레임 구분자 설정("set ifd")은 동시에 사용할 수 없습니다.

프레임 구분자 설정

프레임 구분자를 설정하는 명령어는 ifd입니다. 데이터 수신시 이 명령어로 설정된 구분자를 이용하여 프레임을 구분합니다. 이 때 "get rxlen" 명령으로 프레임 길이를 확인하여 프레임단위로 데이터를 수신할 수 있습니다.

프레임 구분자 설정하기

- 프레임의 끝 구분하기

  프레임의 끝을 구분하기 위해서는 다음과같이 구분자를 설정합니다.

  ```
  "set ifd (del)"
  ```

  del에 프레임 구분자를 설정하면 해당 구분자까지가 하나의 프레임이 됩니다. 구분자는 16진수의 문자열 형태로 설정해야 하며, 최소 2바이트에서 최대 64바이트까지 설정할 수 있습니다.

- 프레임의 처음과 끝을 모두 구분하기

  프레임의 처음과 끝을 모두 구분하기 위해서는 다음과 같이 구분자를 2개 설정합니다.

  ```
  "set ifd (start_del) (end_del)"
  ```

  프레임 구분자를 2개 설정하면 첫 번째 구분자부터 두 번째 구분자까지가 하나의 프레임이 됩니다. 이 때 양 구분자와 구분자 사이의 데이터만 유효하고 나머지 데이터는 무시됩니다. 프레임 구분자 2개의 길이의 합은 64바이트를 초과하여 설정할 수 없습니다.

프레임 구분자 설정 해제하기

프레임 구분자를 설정한 상태에서 이를 해제하는 방법은 다음과 같습니다.

```
"set ifd"
```

위와 같이 ifd명령 뒤에 아무것도 입력하지 않으면 프레임 구분자 설정이 해제됩니다.

- 프레임 구분자 설정 및 해제하기(예)

```
<?php
include "lib/sd_spc.php";

$sid = 14;
```

```
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200");

spc_request_dev($sid, "set ifd 1b01");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output: 1b01

spc_request_dev($sid, "set ifd 1b02 1b03");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output: 1b02 1b03

spc_request_dev($sid, "set ifd");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output:
?>
```

- 프레임 구분자를 이용한 데이터 송/수신 예

  이 예제는 0x0d를 구분자로 하여 프레임 단위로 데이터를 수신하고 해당 데이터를 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");
spc_request_dev($sid, "set ifd 0d");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
```

```
?>
```

※ 주의: 프레임 구분자 설정("set ifd")과 프레임 간격 설정("set ifg")은 동시에 사용할 수 없습니다.

시리얼 전송 지연 설정

이 설정은 PES-2406이 시리얼포트로 데이터를 전송할 때 바이트와 바이트 사이에 지연 시간을 추가하기 위한 설정입니다. 시리얼 전송 지연을 설정하는 명령어는 txdelay입니다.

```
"set txdelay (bits)"
```

bits에 지연 시간을 설정합니다. 설정 단위는 비트이며 0에서 30000까지 설정할 수 있습니다.

- 시리얼 전송 지연 설정 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set txdelay 10");
echo spc_request_dev($sid, "get txdelay"), "₩r₩n"; // output: 10

spc_request_dev($sid, "set txdelay 0");
echo spc_request_dev($sid, "get txdelay");          // output: 0

?>
```

브레이크 신호 전송

브레이크 신호를 전송하는 명령어는 break입니다.

```
"set break (time)"
```

time에 브레이크 시간을 지정합니다.

브레이크 시간은 비트단위 또는 마이크로초 단위로 설정이 가능합니다. 비트단위로 설정할때는 값만 입력하면 되지만 마이크로초 단위로 설정할때는 뒤에 "us"를 붙여야합니다. 설정 가능한 시간은 최소 10비트에서 최대 60초 입니다.

※ 주의: 시스템 부하가 많은 경우 실제 브레이크 시간은 지정한 시간보다 다소 길어질 수 있습니다.

- 브레이크 신호 전송 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set break 10");          // 브레이크 신호 전송: 10비트간격

sleep(1);

spc_request_dev($sid, "set break 1000000us");  // 브레이크 신호 전송: 1초
?>
```

상태 확인하기

PES-2406의 상태를 확인하기 위한 명령어는 get입니다.

통신 파라미터 확인

RS-232의 통신 파라미터를 확인하는 명령어는 uart입니다.

```
"get uart"
```

이 명령어에 의한 응답 형태는 문자열이며 "set uart"의 설정 형식과 동일합니다.

```
"(baudrate)(parity)(data bit)(stop bit)(flow control)"
```

파라미터	응답 범위	설명
baudrate	1200 ~ 115200	통신 속도(bps)
parity	N, E, O, M 또는 S	패리티 비트 (N: 없음, E: 짝수, O: 홀수, M: Mark, S: Space)
data bit	8 또는 7	데이터 비트
stop bit	1 또는 2	정지 비트
flow control	N, H 또는 S	흐름제어 (N: 없음, H: RTS/CTS, S: Xon/Xoff)

- 통신 파라미터 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200");
echo spc_request_dev($sid, "get uart"), "₩r₩n";    // output: 115200N81N

spc_request_dev($sid, "set uart 115200N81");
echo spc_request_dev($sid, "get uart"), "₩r₩n";    // output: 115200N81N

spc_request_dev($sid, "set uart 9600E72H");
echo spc_request_dev($sid, "get uart");             // output: 9600E72H

?>
```

모뎀 회선 신호 확인

모뎀 회선 신호를 확인하는 명령어는 modem입니다. 총 6개의 신호를 동시에 또는 각각 확인할 수 있습니다.

동시에 확인

```
"get modem"
```

이 경우에는 반환값은 6자리의 이진수 형태의 문자열이며 각각의 자리에 대한 의미는 다음과 같습니다.

```
(RI)(CTS)(RTS)(DSR)(DTR)(CD)
```

값 0은 active상태를, 1은 inactive상태를 나타냅니다.

- 동시에 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set modem 11");                  // RTS & DTR: inactive
echo spc_request_dev($sid, "get modem"), "₩r₩n";        // output(e.g.): 111111
sleep(1);

spc_request_dev($sid, "set modem 00");                  // RTS & DTR: active
echo spc_request_dev($sid, "get modem"), "₩r₩n";        // output(e.g.): 110101
sleep(1);
?>
```

각각 확인

```
"get modem (signal)"
```

이 경우에는 signal에 확인하고자하는 신호 이름을 입력합니다.

신호 이름	설명
ri	Ring Indicator

신호 이름	설명
cts	Clear To Send
rts	Request To Send
dsr	Data Set Ready
dtr	Data Terminal Ready
cd	Carrier Detect

- 각각 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set modem rts 1");              // RTS: inactive
echo spc_request_dev($sid, "get modem rts"), "₩r₩n";  // output(e.g.): 1
sleep(1);

spc_request_dev($sid, "set modem dtr 1");              // DTR: inactive
echo spc_request_dev($sid, "get modem dtr");           // output(e.g.): 1
?>
```

내부 카운터 값 확인

내부 카운터 값을 확인하는 명령어는 count입니다.

```
"get count (counter)"
```

counter에 값을 확인할 내부 카운터 이름을 입력합니다.

내부 카운터	설명
rx	수신 바이트 카운터
tx	송신 바이트 카운터
rf	수신 프레임 카운터
tf	송신 프레임 카운터
pe	패리티 에러 카운터
fe	프레이밍 에러 카운터
oe	오버런 에러 카운터
be	브레이크 에러 카운터
rbo	수신 버퍼 오버플로우 카운터
tbo	송신 버퍼 오버플로우 카운터
rfo	수신 프레임 포인터 오버플로우 카운터
tfo	수신 프레임 포인터 오버플로우 카운터

- 내부 카운터 값 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

echo spc_request_dev($sid, "get count rx"), "₩r₩n";
echo spc_request_dev($sid, "get count tx"), "₩r₩n";
echo spc_request_dev($sid, "get count rf"), "₩r₩n";
echo spc_request_dev($sid, "get count tf"), "₩r₩n";
echo spc_request_dev($sid, "get count pe"), "₩r₩n";
echo spc_request_dev($sid, "get count fe"), "₩r₩n";
echo spc_request_dev($sid, "get count oe"), "₩r₩n";
echo spc_request_dev($sid, "get count be"), "₩r₩n";
echo spc_request_dev($sid, "get count rbo"), "₩r₩n";
echo spc_request_dev($sid, "get count tbo"), "₩r₩n";
echo spc_request_dev($sid, "get count rfo"), "₩r₩n";
echo spc_request_dev($sid, "get count tfo");
?>
```

프레임 간격 확인

프레임 간격을 확인하는 명령어는 ifg입니다.

```
"get ifg"
```

응답 값의 단위는 비트(bits)입니다.

- 프레임 간격 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set ifg 50");
echo spc_request_dev($sid, "get ifg"), "₩r₩n";

spc_request_dev($sid, "set ifg 0");
echo spc_request_dev($sid, "get ifg");
?>
```

프레임 구분자 확인

프레임 구분자를 확인하는 명령어는 ifd입니다.

```
"get ifd"
```

응답 값은 다음과 같은 형태의 16진수의 문자열 형식입니다.

```
[start_del [end_del]]
```

- 프레임 구분자 확인 예

```
<?php
include "lib/sd_spc.php";

$sid = 14;

spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200");

spc_request_dev($sid, "set ifd 1b01");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output: 1b01

spc_request_dev($sid, "set ifd 1b02 1b03");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output: 1b02 1b03

spc_request_dev($sid, "set ifd");
echo spc_request_dev($sid, "get ifd"), "₩r₩n";  // output:
?>
```

시리얼 전송 지연 확인

시리얼 전송 지연을 확인하는 명령어는 txdelay입니다.

```
"get txdelay"
```

응답값의 단위는 비트입니다.

- 시리얼 전송 지연 확인 예

```
<?php
include "/lib/sd_spc.php";

$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

spc_request_dev($sid, "set txdelay 10");
echo spc_request_dev($sid, "get txdelay"), "₩r₩n"; // output: 10

spc_request_dev($sid, "set txdelay 0");
echo spc_request_dev($sid, "get txdelay");          // output: 0

?>
```

수신 데이터 크기 확인

수신 데이터 크기를 확인하는 명령어는 rxlen입니다. 프레임 간격("set ifg") 또는 프레임 구분자("set ifd")가 설정된 경우에는 수신된 프레임의 길이가 반환됩니다. 이 때 수신된 프레임이 여러개인 경우 가장 먼저 수신된 프레임의 길이가 반환됩니다.

```
"get rxlen [del]"
```

응답 값은 정수 형태의 문자열입니다. del에 구분자를 지정하면 해당 구분자까지의 길이를 반환합니다.

※ 구분자 설정은 프레임 구분자 설정의 "프레임의 끝 구분하기"를 참조하시기 바랍니다.

- 수신 데이터 크기 확인 예

  이 예제는 시리얼포트로 수신한 데이터를 그대로 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
```

```
?>
```

- 지정한 구분자까지의 수신 데이터 크기 확인 예

  이 예제는 0x0d를 구분자로 하여 프레임 단위로 데이터를 수신하고 해당 데이터를 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen 0d");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
?>
```

수신 버퍼 크기 확인

수신 버퍼의 크기를 확인하는 명령어는 rxbuf입니다.

```
"get rxbuf"
```

응답 값은 정수 형태의 문자열입니다.

- 수신 버퍼 크기 확인 예

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

$rxbuf_len = (int)spc_request_dev($sid, "get rxbuf");
echo $rxbuf_len;     // output(e.g.): 12288
?>
```

송신 버퍼 여유공간 확인

송신 버퍼 여유공간을 확인하는 명령어는 txfree입니다.

```
"get txfree"
```

응답 값은 정수 형태의 문자열입니다.

- 송신 버퍼 여유공간 확인 예

  이 예제는 시리얼포트로 수신한 데이터를 그대로 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
?>
```

송신 버퍼 크기 확인

송신 버퍼의 크기를 확인하는 명령어는 txbuf입니다.

```
"get txbuf"
```

응답 값은 정수 형태의 문자열입니다.

- 송신 버퍼 크기 확인 예

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

$txbuf_len = (int)spc_request_dev($sid, "get txbuf");
echo $txbuf_len;     // output(e.g.): 12288
?>
```

데이터 수신하기

데이터를 수신하기 위해서는 spc_request함수를 사용합니다. 이 때 두 번째 인자에 6을 입력해야 합니다.

```
$rbuf = spc_request($sid, 6, $rlen)
```

- $rbuf: 수신 데이터를 저장할 버퍼
- $sid: 보드에 설정된 슬레이브 아이디
- $rlen: 수신할 바이트 수

데이터 수신 예

이 예제는 시리얼포트로 수신한 데이터를 그대로 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
?>
```

데이터 송신하기

데이터를 송신하기 위해서는 spc_request함수를 사용합니다. 이 때 두 번째 인자에 7을 입력해야 합니다.

```
spc_request($sid, 7, $wbuf)
```

- $sid: 보드에 설정된 슬레이브 아이디
- $wbuf: 송신할 데이터

데이터 송신 예

이 예제는 시리얼포트로 수신한 데이터를 그대로 다시 송신하는 예제입니다.

```
<?php
include "/lib/sd_spc.php";

$rwbuf = "";
$sid = 14;
spc_reset();
spc_sync_baud(115200);

spc_request_dev($sid, "set uart 115200N81");

while(1)
{
    $txfree = (int)spc_request_dev($sid, "get txfree");
    $rlen = (int)spc_request_dev($sid, "get rxlen");
    if($rlen > 0)
    {
        if($rlen <= $txfree)
        {
            // receive data
            $rwbuf = spc_request($sid, 6, "$rlen");

            // send data
            spc_request($sid, 7, $rwbuf);

            // print data
            echo $rwbuf;
        }
    }
    usleep(1000);
}
?>
```