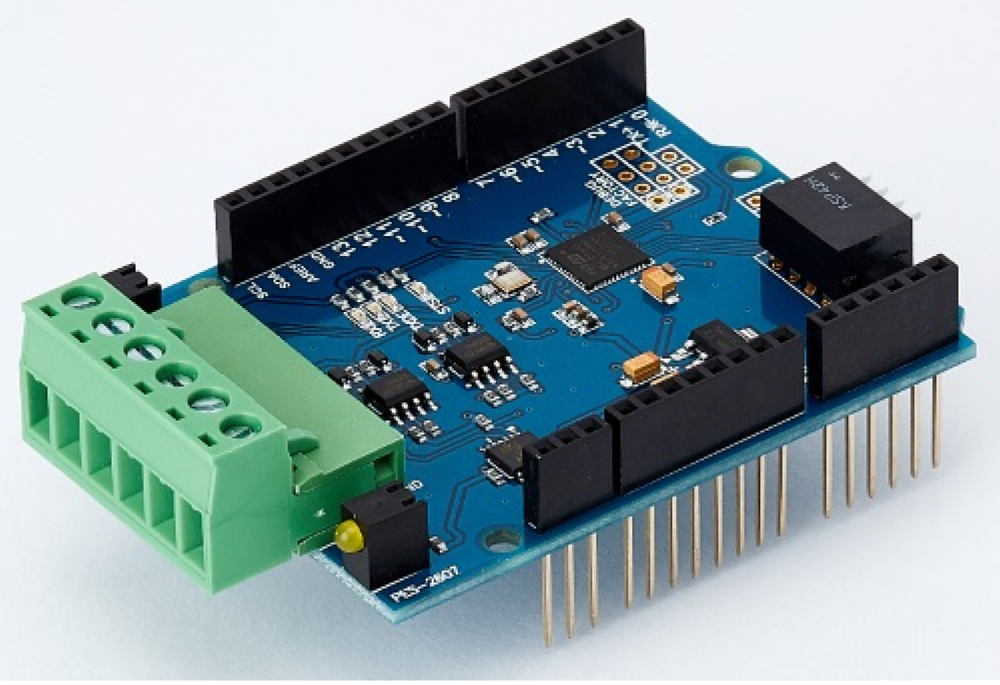
Introduction



PES-2607

PES-2607, smart RS-422/485 board, is one of smart expansion boards for PHPoC Shields for Arduino. You can easily implement the RS-422 or RS-485 communication on the Arduino board by using this board and a PHPoC shield.

Highlights of PES-2607

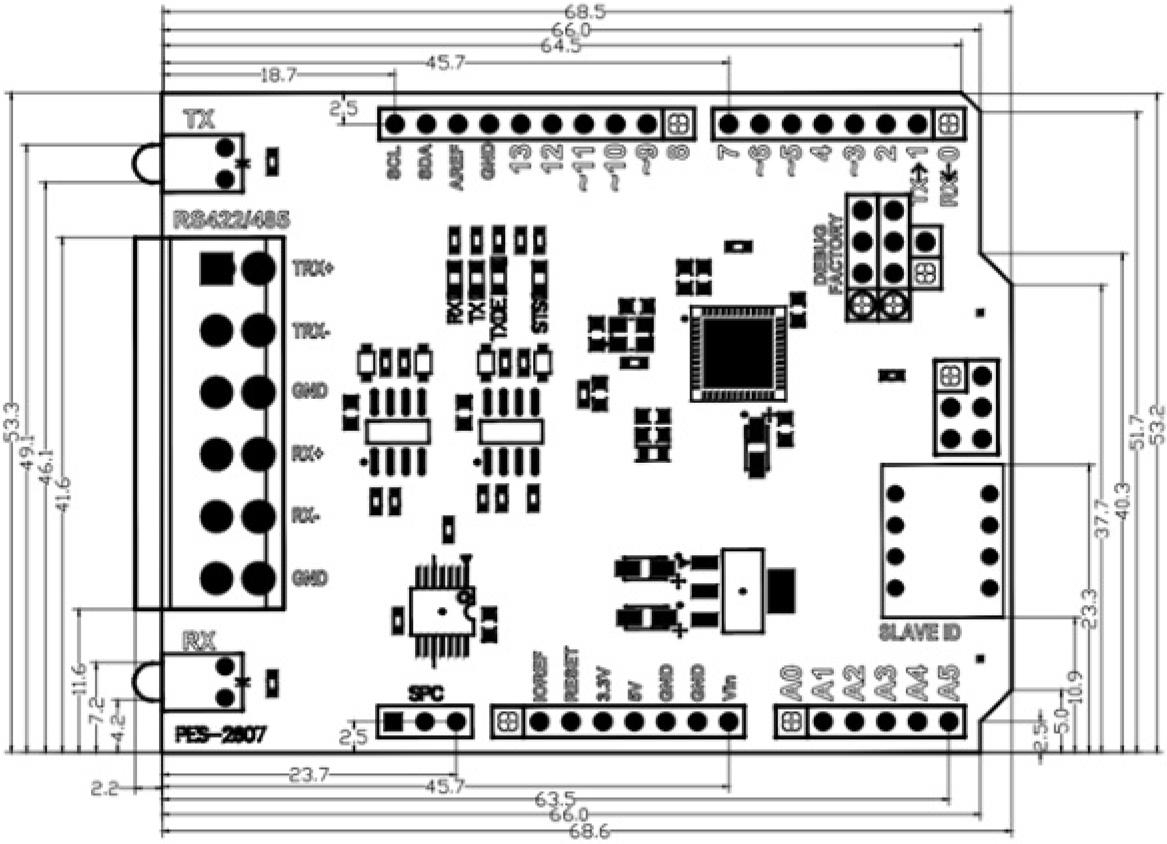
- 1 X RS-422(or RS-485) port : 1200bps ~ 115200bps

※ Caution : Both a PHPoC shield with R2 or later version and an Arduino board are required to use this board!

> What is the Smart Expansion Board for PHPoC shield?
>
> A smart expansion board for PHPoC shield has own devices and firmware. This board communicate with a PHPoC shield in a master-slave protocol through the designated port. Two or more smart expansion boards can be connected to one PHPoC shield and each of them required to be setting a slave id.

Dimension

Body



PES-2607 Dimension (mm)

※ Dimensions(unit : mm) may vary according to a method of measurement.

Terminal Block

This board uses two types of 6-pole terminal block. Refer to each datasheet for dimension.

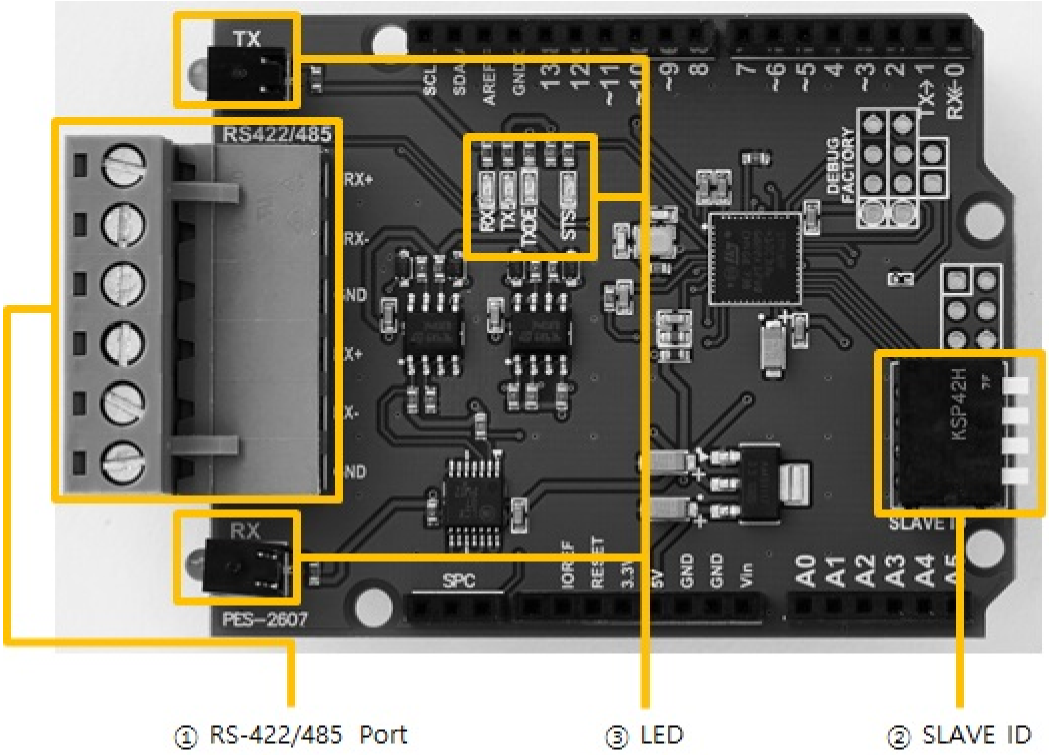
- Datasheet of T-type Terminal Block
- Datasheet of S-type Terminal Block

Schematic

This is the schematic of PES-2607.

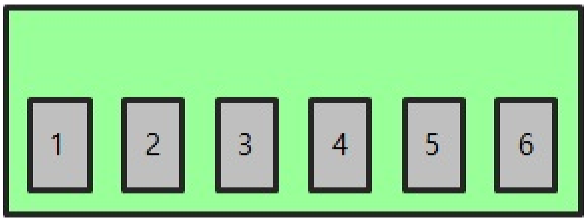
- PES-2607-V10-PO.pdf

Layout



1. RS-422/485 Port

The RS-422/485 port of this board is a terminal block. It has six terminals and a 5mm pitch.



RS-422

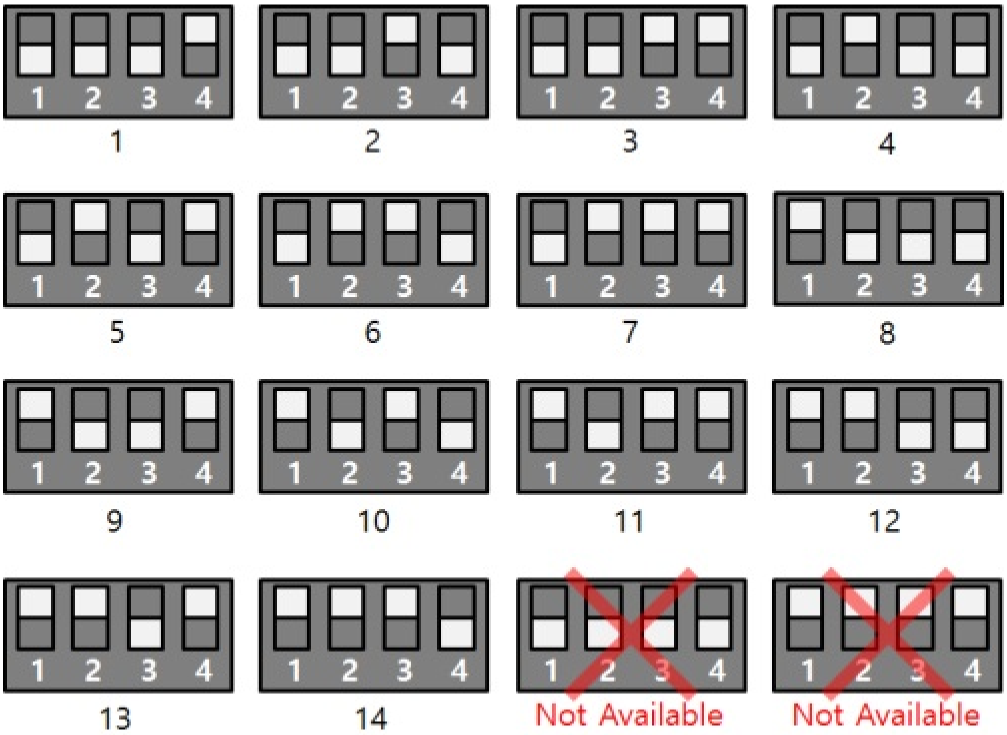
Number	Name	Description	Level	I/O	Wiring
1	TX+	Transmit Data+	RS-422	Out	Required
2	TX-	Transmit Data-	RS-422	Out	Required
3	GND	Ground	Ground	-	Required
4	RX+	Receive Data+	RS-422	In	Required
5	RX-	Receive Data-	RS-422	In	Required
6	GND	Ground	Ground	-	Required

RS-485

Number	Name	Description	Level	I/O	Wiring
1	TRX+	Transmit / Receive Data+	RS-485	In/Out	Required
2	TRX-	Transmit / Receive Data-	RS-485	In/Out	Required
3	GND	Ground	Ground	-	Required
6	GND	Ground	Ground	-	Required

2. SLAVE ID Switch

A slave ID is used when PHPoC board identifies each smart expansion board. So, each smart expansion board, which is connected to a PHPoC board, should have a unique slave ID. The slave ID can be set one of the numbers from 1 to 14 by 4 DIP switches as follows:



3. LED

This board has 6 LEDs.

Name	Quantity	Type	Color	Operation
STS	1	SMD	red	ID setting is normal > repeats on / off every 1 second ID setting is incorrect > blinks fast
TX	2	DIP, SMD	green	blinks when sending data to the serial port
RX	2	DIP, SMD	yellow	blinks when receiving data to the serial port
TXDE	1	SMD	green	TxDE enabled > blinks while sending data TxDE disabled > ON

How to Use

This board can be used by steps as follows.

1. Connect to a PHPoC Shield and an Arduino

It is not possible to use this board alone. Please be sure that connection to a PHPoC Shield and an Arduino.

2. Install Libraries for Arduino

Install PHPoC and PhpocExpansion library via library manager on Arduino IDE. Both libraries are required to use PHPoC shield and this board. Refer to the manual pages below for detail about the libraries.

- PHPoC shield library reference

3. Use Sample Codes

Use sample codes in libraries and examples in this manual.

Class and Functions

Class

To use this extension board, use the ExpansionSerial class of the PHPoC Expansion library.

Member Functions

Available member functions of the ExpansionSerial class are as follows:

Member Function	Description
int getPID(void)	get the product's ID
char *getName(void)	get the product's name
ExpansionSerial(int sid)	create an instance of the serial port
void begin(void)	set the serial communication parameters
int available(void)	get the received data size
int peek(void)	peek one byte of received data
int read(void)	read one byte from the receive buffer
int availableForWrite(void)	get the remaining size of receive buffer
void flush(void)	flush send buffer
int write(int wbuf, int wlen)	send data

Settings

Setting Communication Parameters

You can set communication parameters by using begin() function.

```
port.begin(baud)
port.begin(sets)
```

- baud - baudrate in bps unit (1200 ~ 115200)
- sets - a string which is specified baudrate, parity, data bit, stop bit and flow control

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

※ (): mandatory, []: optional

Parameter	Values	Description	Default Value
baudrate	1200 ~ 115200	baudrate(bps)	115200
parity	N, E, O, M or S	parity bit (N:None, E:Even, O:Odd, M:Mark, S:Space)	N
data bit	8 or 7	data bit	8
stop bit	1 or 2	stop bit	1
flow control	T or N	TxDE enable (T) / disable (S)	T

※ Note : It is recommended that you always set the flow control to T.

Example

- source code for Arduino

```
#include <PhpocExpansion.h>
#include <Phpoc.h>
#define BUFFER_SIZE 100  // read and write buffer size, reduce it if memory of Arduino is not enough

byte spcId = 1;

ExpansionSerial port(spcId);

byte rwbuf[BUFFER_SIZE];  // read and write buffer

void setup() {
    Serial.begin(9600);
    while(!Serial)
        ;
    Phpoc.begin(PF_LOG_SPI | PF_LOG_NET);
    Expansion.begin();

    // sets the parameters for serial data communication
    port.begin("115200N81T");
}

void loop() {
    int txfree = port.availableForWrite();
    int rxlen = port.available();

    if(rxlen > 0) {
        if(rxlen <= txfree) {
            int rwlen; // read and write length

            if(rxlen <= BUFFER_SIZE)
                rwlen = rxlen;
            else
                rwlen = BUFFER_SIZE;

            // receive data
            rwlen = port.readBytes(rwbuf, rwlen);

            // send data
            port.write(rwbuf, rwlen);

            // print data to serial monitor of Arduino IDE
            Serial.write(rwbuf, rwlen);
        }
    }

    delay(1);
}
```

Receiving Data

Getting Received Data Size

You can get the received data size from the serial port by using available() function.

```
port.available()
```

This function returns the data size (bytes in integer) which can be read from the serial port.

Peeking a Byte

You can peek the first byte in the receive buffer by using peek() function.

```
port.peek()
```

The byte returned by this function remains in the buffer.

Reading a Byte

You can read the first byte in the receive buffer by using read() function.

```
port.read()
```

The byte returned by this function removed from the buffer.

Example

- source code for Arduino

```
#include <PhpocExpansion.h>
#include <Phpoc.h>
#define BUFFER_SIZE 100  // read and write buffer size, reduce it if memory of Arduino is not enough

byte spcId = 1;

ExpansionSerial port(spcId);

byte rwbuf[BUFFER_SIZE];  // read and write buffer

void setup() {
    Serial.begin(9600);
    while(!Serial)
        ;

    Phpoc.begin(PF_LOG_SPI | PF_LOG_NET);
    Expansion.begin();
    port.begin("115200N81T");
}

void loop() {
    int txfree = port.availableForWrite();

    // gets the size of received data
    int rxlen = port.available();

    if(rxlen > 0) {

        // reads the next byte of incoming serial data
        int value = port.read();
        Serial.print("read : ");
        Serial.println(value);

    }
    delay(1);
}
```

Sending Data

Getting the Free Space in Send Buffer

You can get the free space in send buffer by using availableForWrite() function.

```
port.availableForWrite()
```

This function returns the size of free space (byte in integer) in send buffer.

Flushing the Send Buffer

You can flush the send buffer by using flush() function.

```
port.flush()
```

Sending Data

You can send data by using write() function.

```
port.write(byte)
port.write(wbuf, wlen)
```

- byte - one-byte data in integer
- wbuf - a series of bytes
- wlen - the size of send data (bytes)

Example

- source code for Arduino

```
#include <PhpocExpansion.h>
#include <Phpoc.h>
#define BUFFER_SIZE 100  // read and write buffer size, reduce it if memory of Arduino is not enough

byte spcId = 1;

ExpansionSerial port(spcId);

byte rwbuf[BUFFER_SIZE];  // read and write buffer

void setup() {
    Serial.begin(9600);
    while(!Serial)
        ;

    Phpoc.begin(PF_LOG_SPI | PF_LOG_NET);
    Expansion.begin();
    port.begin("115200N81T");
}

void loop() {

    int txfree = port.availableForWrite();
    int rxlen = port.available();

    if(rxlen > 0) {
        if(rxlen <= txfree) {
            int rwlen; // read and write length

            if(rxlen <= BUFFER_SIZE)
                rwlen = rxlen;
            else
                rwlen = BUFFER_SIZE;

            // receive data
            rwlen = port.readBytes(rwbuf, rwlen);

            // send data
            port.write(rwbuf, rwlen);

            // print data to serial monitor of Arduino IDE
            Serial.write(rwbuf, rwlen);
        }
    }

    delay(1);
}
```