periCORE Development Kit

Full Featured Firmware Development Setup

User Guide

Perinet
Seamless IoT Connectivity

Doc-No.: PRN.100.378 rev: 6

2026-05-04

Document Information

Title	periCORE Development Kit
Subtitle	Full Featured Firmware Development Setup
Type	User Guide
Status	Release
Version	6
Date	2026-05-04
Disclosure Restriction	

Contents

1 Overview

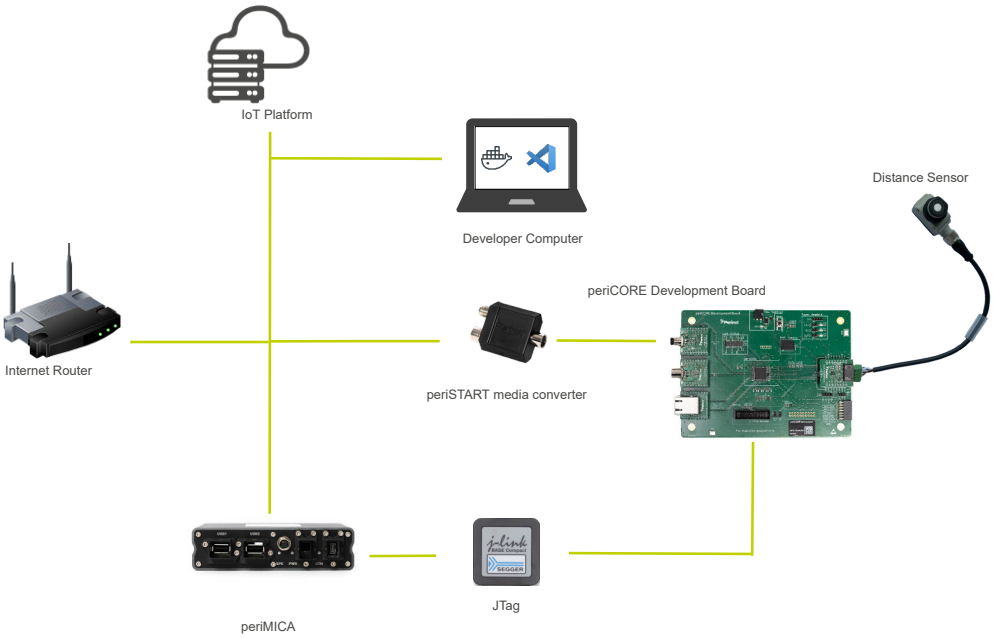


Figure 1: Development Kit Overview

The *periCORE development kit* is a set of tools used to build and test custom IIoT sensor and actuator applications that can be implemented with the *periCORE SPE communication module*. Its purpose is to provide an IIoT environment that allows new applications to be implemented efficiently. Various features for secure communication are provided as a software library. Figure 2 shows the firmware development effort for an example application (periNODE-distance-firmware [9]). The majority of the implementation effort has already been provided by Perinet.

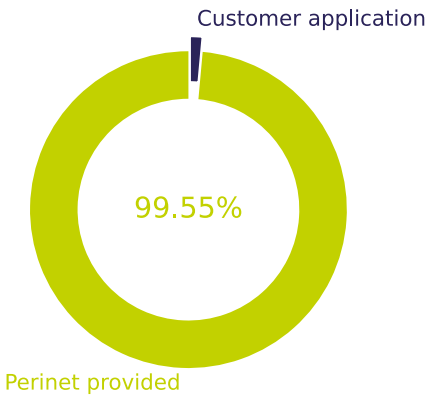


Figure 2: Distribution of development efforts for a periCORE-based firmware.

The first section (Section 2) presents the *periCORE Development Board* hardware architecture.

All the interfaces available in the module can be accessed via the development board peripherals.

The next section (Section 3) guides the user through the configuration and use of the provided software tools. It covers the *periCORE SDK* and the *pericoredbg Container*, which are used to build and debug customized *periCORE*-based applications without requiring any additional software to be installed — a significant benefit for developers.

Information on how to implement applications for *periCORE*-based devices is given in Section 4. It focuses on how to extend the network services provided by *libperiCORE* and how to customize the Web UI frontend.

Another powerful tool relevant to the kit is the *periMICA*, which can run typical field IIoT functions through its lightweight containers, e.g. the *MQTT Broker Container* (available for download at https://downloads.perinet.io).

Targeted Applications

- Industrial sensors
- Industrial control
- IoT / IIoT
- Remote sensor access
- Building automation

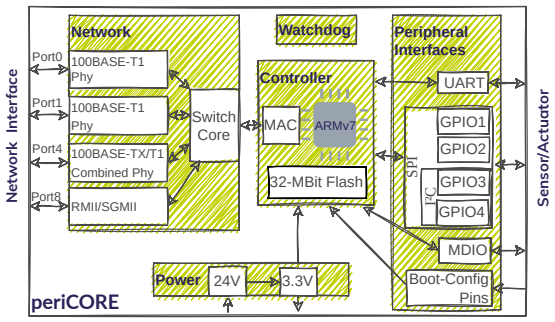


Figure 3: periCORE hardware blocks.

Key Features

- Fully qualified Industrial IoT module
- Firmware development framework
- Provided TCP/IPv6 stack
- Event-based minimal operating system
- arm Cortex®-R4 250MHz processor core
- 32-MBit flash memory for persistent storage
- Up to 3x 100BASE-T1 Single Pair Ethernet Phys (IEEE 802.3bw compatible)
- Integrated Ethernet switching core
- Compact form factor
- Operated with 24V
- Integrated 3V3 power supply
- patent-protected

Operational Parameters

- Operating voltage: 24 VDC
- Power supply: 3.3 VDC (up to 100mA)
- Temperature range: -40°C to +85°C
- Power consumption: 0.6 W

Package

Dimensions: 16.7 x 13 x 3.9 mm (Figure 4)

Mounting: Solder pads, 73 LGA-Pads, Pattern 13 x 10, Pitch 1.27 mm

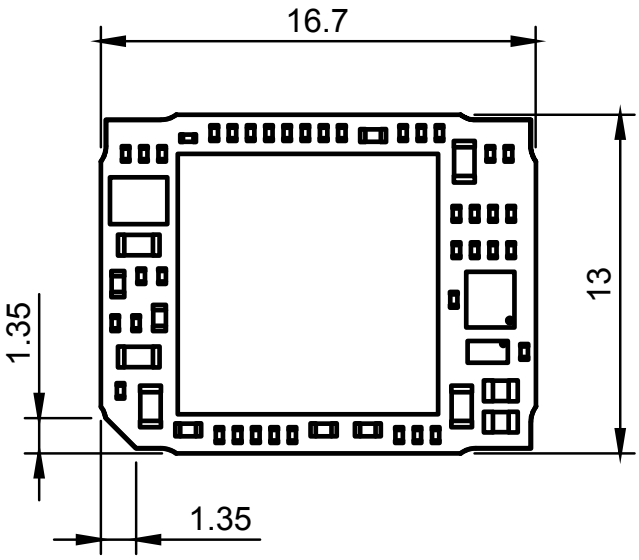


Figure 4: periCORE dimensions in mm.

Interfaces

- 2 x 100BASE-T1 Phy (IEEE 802.3bw)
- 1 x Combined 100BASE-T1/TX Phy
- 1 x RMII | SGMII
- 1 x MAC to arm processor core (Figure 3)
- 1 x UART
- 1 x I2C (400 kHz ± 20 %)
- 2 x GPIO
- 1 x optional SPI (800 kHz ± 20 %)

Compliance

- CRA ready

- RoHS 3 (EU 2015/863)
- WEEE (2012/19/EU)
- REACH

Security

- NIST compliant TLS implementation
- Role Based Access Control (RBAC)
- Certificate based client authentication
- AES encryption algorithm
- X.509 certificates & PKIX path validation
- Elliptic Curve Cryptography (ECC)

Software Library *libperiCORE*

- Rapid firmware development with *periCORE Development Kit* (see Figure 5)
- mDNS/LLMNR for name resolving
- DNS-SD for automated service discovery
- TCP/UDP endpoints
- TLS-based secure communication endpoints
- RESTful API
- Secure MQTT-client for publishing sensor values or subscribing to actuator commands
- HTTPs server including Web based UI
- Product lifecycle features
- C++20 standard conform

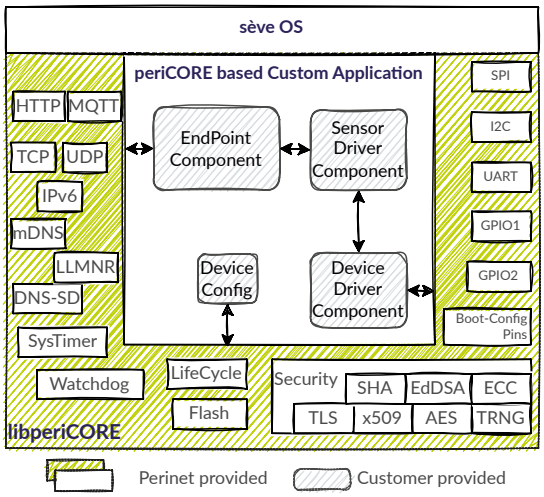


Figure 5: The software architecture with Custom Application template, provided by Perinet.

2 Hardware Architecture

The *periCORE SPE communication module* is an electronic device that provides sensor communication and interaction via Single Pair Ethernet interface. The *periCORE Development Board* is intended to simplify the development of *periCORE* based applications. The hardware design of the development board is kept modular from both network and sensor sides in order to support a wide range of applications.

In this case, modularity means that the interfaces (connectors as well as supporting circuits) for either network and sensor side are provided by a specific daughterboard. This allows the user to evaluate multiple different network and sensor interfaces on the *periCORE Development Board*.

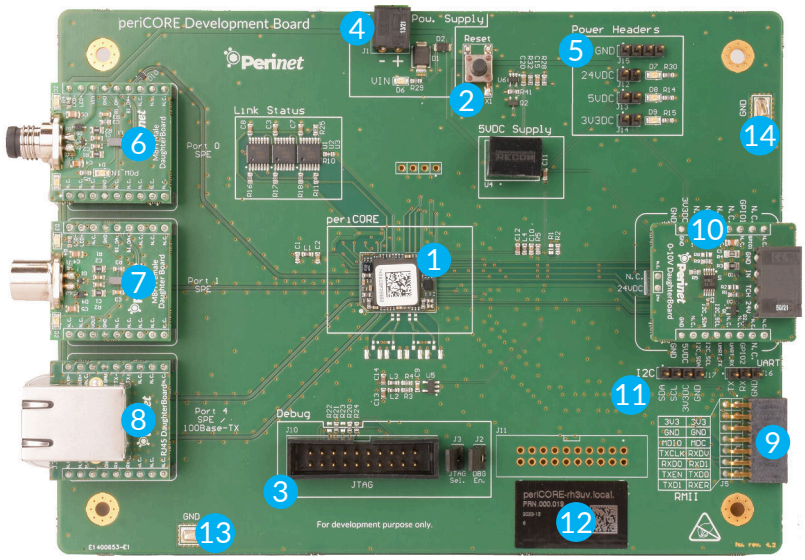


Figure 6: Top view on *periCORE Development Board* with all jumpers in default position, including all daughterboards.

The relevant parts of the board are (Figure 6):

1. *periCORE* module
2. Reset button — used to reset the *periCORE* module
3. JTAG connector (`J10`) for debug purposes with standard ARM pinout ([2])
4. Power supply connector (`J1`)
5. Power header connectors (`J12`, `J13`, `J14` and `J15`) with different voltage levels (24V, 5V, 3.3V and GND)
6. Port 0 (SPE) with M8H Male daughterboard attached
7. Port 1 (SPE) with M8H Female daughterboard attached
8. Port 4 (SPE / 100BASE-TX) with RJ45 daughterboard attached
9. RMII header connector for connecting with external `PHYs`

10. Sensor daughterboard (with 0-10 Daughterboard attached)
11. Header connectors J16 and J17 for accessing UART and I2C signals
12. *periCORE* device label with information such as mDNS name, production date and the version number
13. GND clamps for connecting measuring instruments

The schematic of the *periCORE Development Board* are given in Appendix A.

2.1 Power Supply

The nominal supply voltage for the development board is 24V DC which is converted to 3.3V inside the *periCORE* module. On the development board, an additional voltage regulator (U4) is used for generating 5V. There are three LEDs (D7, D8, and D9) which are indicating the presence of 24V, 5V, and 3.3V, respectively. Power to the *periCORE* development board can be supplied in two ways:

- via J1 PCB header connector
- over the network daughterboard via a hybrid cable

2.1.1 Power over J1 connector

J1 is a male PCB header connector with a 3.5 mm pitch. A recommended mating counterpart is har-flexicon 3,50 FSH-2 GN from HARTING (Part number: 14 31 021 4402 000). LED D6 indicates the presence of the voltage on J1. The supply voltage is 24V DC and the polarity of the connector is shown on the board overlay.

2.1.2 Power over a network daughterboard

The concept of hybrid cabling assumes that both power and network signals are transferred over a single cable. In the case of SPE, the hybrid cables have 4 wires (2 pairs of wires). One pair is for the power supply, and the other pair is for SPE. The footprint of the network daughterboard (Figure 21) allows power to flow into or out of the development board. If the power for the development board is provided by the network communication daughterboard, the LED indicating the presence of the input voltage is implemented on the daughterboard itself. The following network daughterboards are relevant for hybrid cabling:

- M8H Male DaughterBoard — power flows into the development board (more information in Section B.1)
- M8H Female DaughterBoard — power flows out of the development board (more information in Section B.2)
- T1H DaughterBoard — power can flow in either direction depending on the wiring (more information in Section B.4)

More information about the network communication daughterboards can be found in Appendix B.

2.1.3 Reset button

Pressing the Reset button (X1) performs a reset of the *periCORE* module.

2.2 Network Interfaces

periCORE has 4 network ports and supports the following interfaces:

- Ethernet MDIs:
  - 100BASE-T1 – Ethernet over a single twisted pair (SPE)
  - 100BASE-TX – Ethernet over two twisted pairs
- RMII

The overview of the network ports and supported interfaces is given in table Table 1:

Port	100BASE-T1	100BASE-TX	RMII
0	X		
1	X		
4	X	X	
8			X

Table 1: *periCORE* supported network interfaces

2.2.1 Ethernet MDIs

There are 3 ports for 100BASE-T1 and 100BASE-TX:PORT0, PORT1, and PORT4. PORT0 and PORT1 support only 100BASE-T1 (SPE), while PORT4 supports both 100BASE-T1 and 100BASE-TX, but only one at a time. The aforementioned ports are available on the development board in the form of slots where the user can place a daughterboard, as shown in Figure 6 under designators 6, 7 and 8. Appendix B has more details about the network daughterboards.

One of the main features of the *periCORE* is the ability to daisy-chain sensor nodes from *Perinet Smart Components*, i.e. other *periCORE*-based devices. Power and communication lines arrive at one node and pass on from that node to the next, and so on. Development and prototyping of such applications is possible with this development board because the footprint of the network daughterboard supports both power input and output. In a typical daisy-chain application the power is supplied to the development board via the M8H Male DaughterBoard, while the M8H Female DaughterBoard is used to supply power to the next node in the chain.

2.2.2 RMII

periCORE development board supports connecting external PHYs via its header connector J5, as shown in Figure 6, marked with number 9. The pinout of the connector is printed on the development board overlay and also shown in Figure 7.

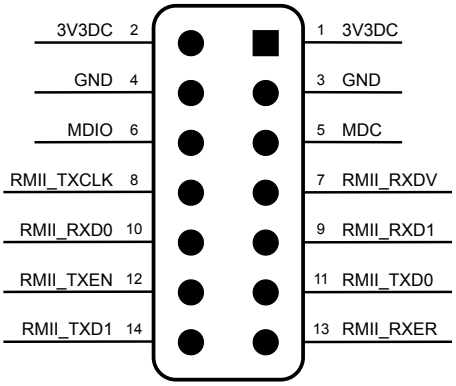


Figure 7: RMII header connector pinout

2.3 Sensor/actuator Interface

For interfacing sensors and actuators, the *periCORE* module is equipped with 2 GPIO pins, I^2C and UART. These interfaces are available on the *periCORE development board* in the form of a slot for placing sensor/actuator daughterboards. The position of the slot is marked with number 10 in Figure 6. The footprint of the sensor/actuator daughterboards is partly compatible with the mikroBUS™ standard [15]. Compared to the standard, there is no support for SPI or analog signals, and the footprint has two additional pins, one of which is for `24V`. The pinout of the sensor/actuator daughterboard slot on the development board is shown below:

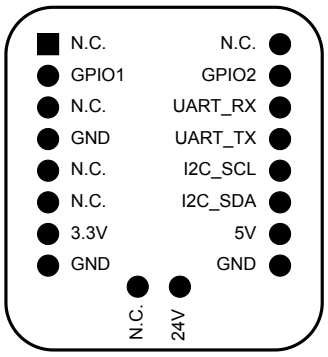


Figure 8: Sensor/actuator daughterboard pinout.

Additionally, UART and I^2C signals are available on the pin headers `J16` and `J17`, respectively.

The *periCORE Development Board* comes with the following sensor/actuator daughterboards:

- Perinet *0-10V DaughterBoard* - for development of 0-10V sensor applications. It is based on the ADS1113 ADC from TI.
- Perinet *pT100 DaughterBoard* - for development of the sensor applications based on pT100 temperature probes. It is based on the ADS122C04 ADC from TI.

- Perinet *GPIO Daughterboard* - for development of the 24V digital I/O based on the MAX22515 ICs.
- mikroE *SHT click* board [1] - for development of the applications based on the humidity and temperature sensor SHT31-DIS from Sensirion.

More details about the Perinet daughterboards, as well as their schematics are given in Appendix C.

2.4 Debug interface

The *periCORE Development Board* allows debugging via JTAG interface. The JTAG debugger can be connected with the development board via the header connector `J10` (see again Figure 6 number 7). The pinout of `J10` is compatible with ARM JTAG 20. Jumpers `J2` and `J3` should be placed in order to have proper JTAG operation.

2.5 Boot-Config Pins

The *periCORE* module exposes three boot-configuration straps, `BOOT_CONFIG_PIN_1`, `BOOT_CONFIG_PIN_2` and `BOOT_CONFIG_PIN_3`, that are sampled at every power-on reset and select the boot mode of the module. Refer to the *periCORE* datasheet [10] for the electrical and timing characteristics of these pins.

On the *periCORE Development Board* the three pins are routed to test points so that they can be driven low or held at their internal pull-up level during the power-up sequence. Their positions on the board are shown in Figure 9.

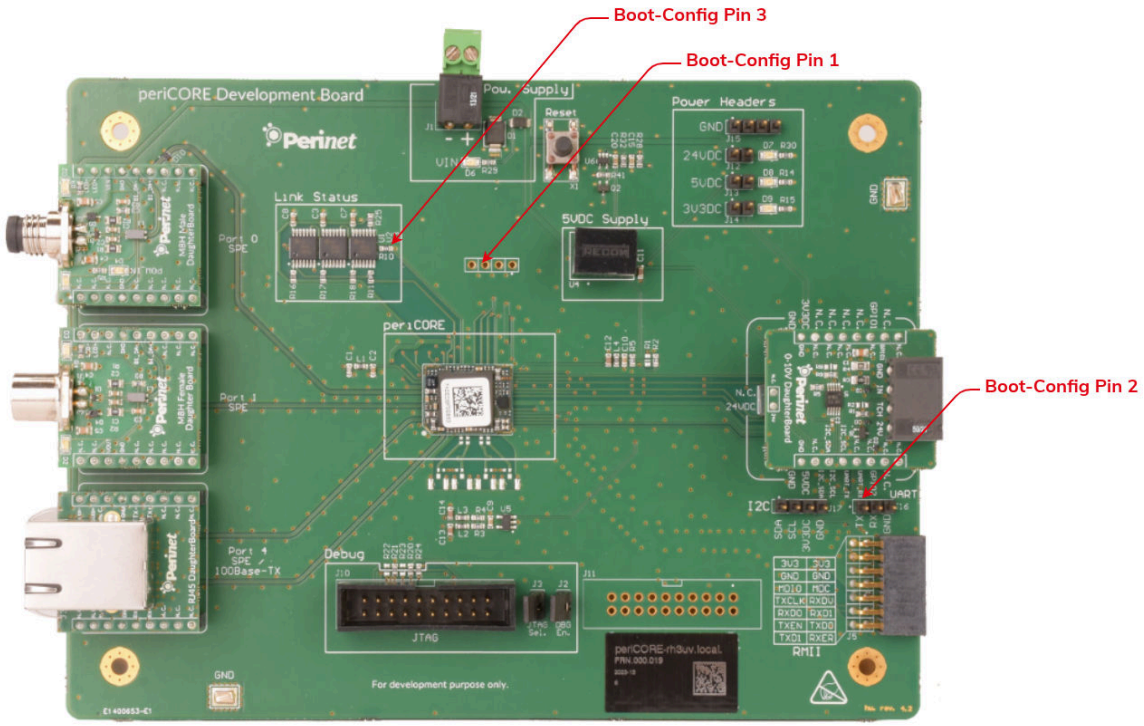


Figure 9: Location of the boot-config pin test points on the *periCORE Development Board*.

The Boot-Config Pins in Figure 9 are:

1. `BOOT_CONFIG_PIN_1` – routed from *periCORE* pad `K10`
2. `BOOT_CONFIG_PIN_2` – routed from *periCORE* pad `N8`, also available on the UART/I2C header cluster (`J16`/`J17`, see Figure 6 number 11)
3. `BOOT_CONFIG_PIN_3` – routed from *periCORE* pad `F10`

The function of the Boot-Config Pins is defined by the application firmware. Their values are evaluated only during the power-up/reset window specified in the *periCORE* datasheet [10]; outside that window the pins resume their normal peripheral function.

Note: To access `BOOT_CONFIG_PIN_1` via the connector, the unpopulated series resistors (see Figure 10) must be soldered.

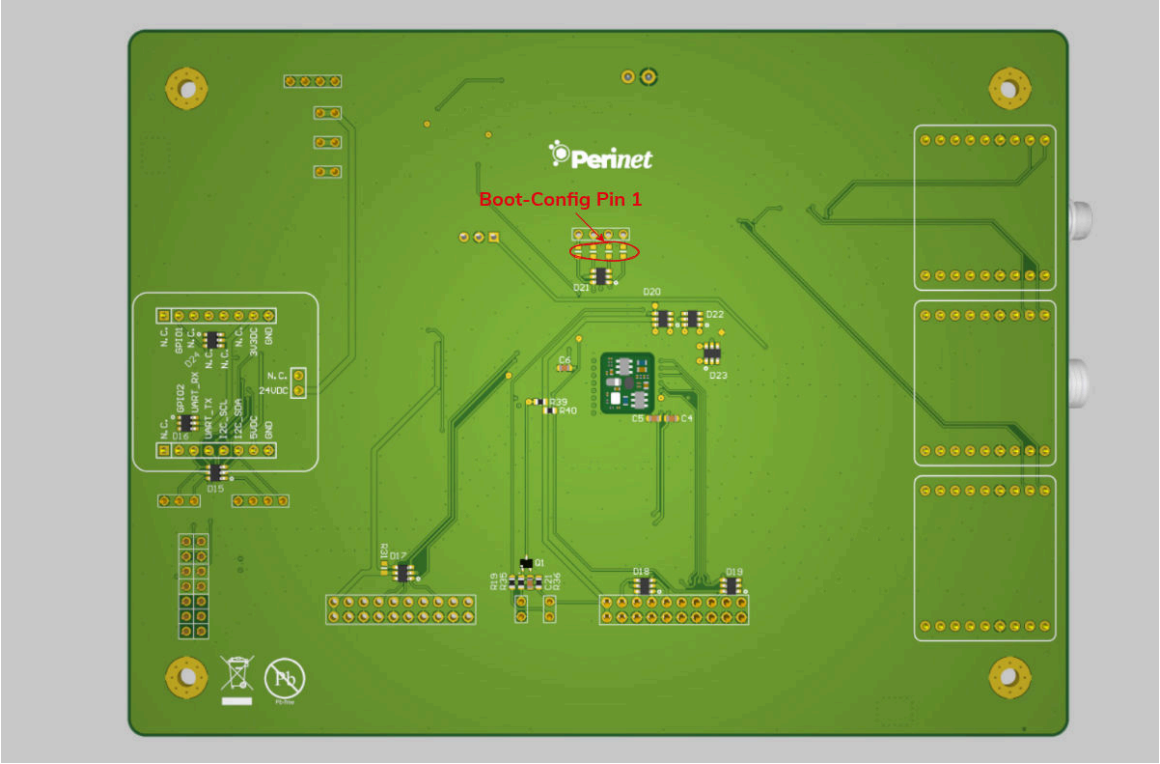


Figure 10: Missing resistors for `BOOT_CONFIG_PIN_1` on the bottom side of the *periCORE Development Board*.

3 System Architecture

3.1 Overview

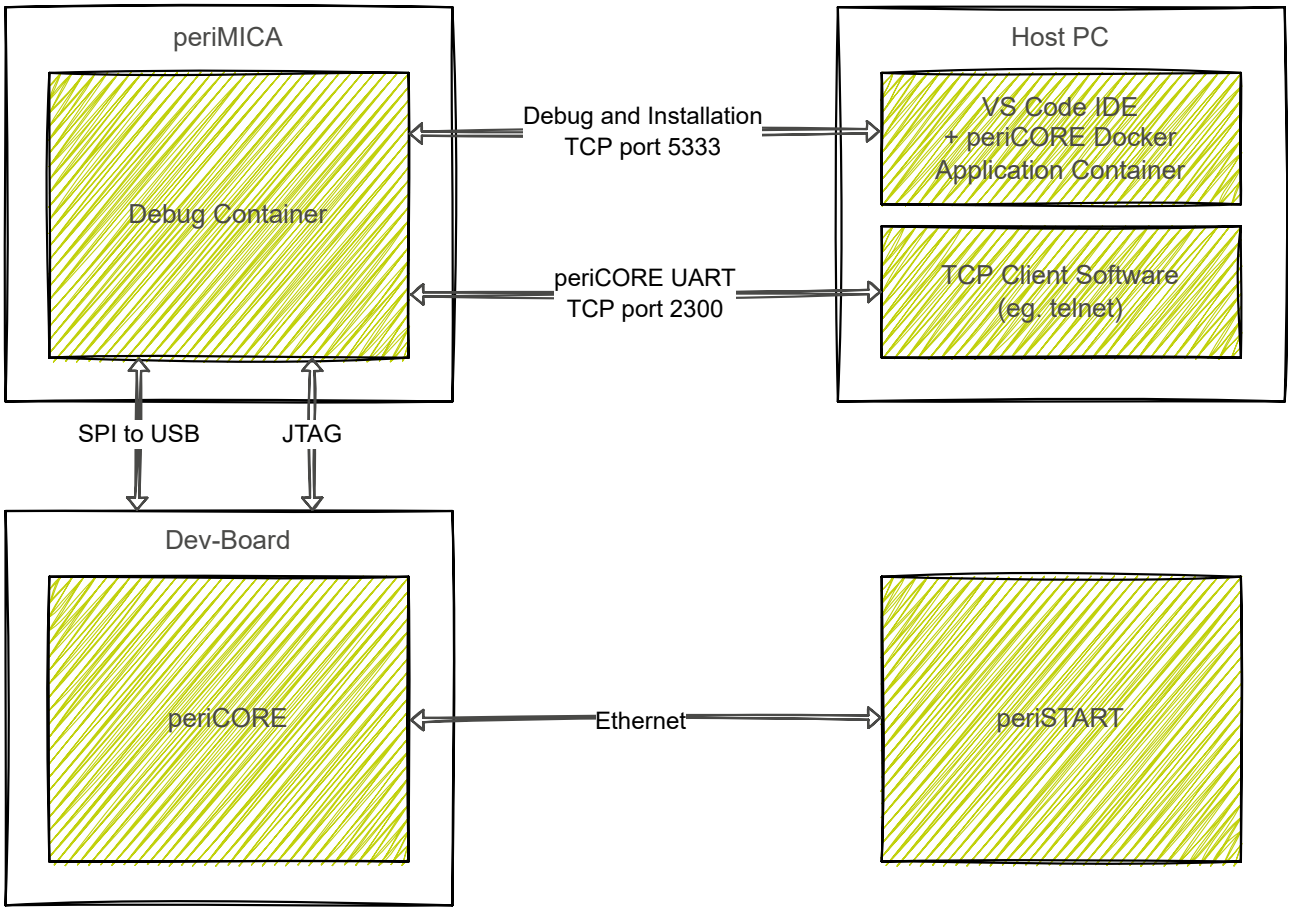


Figure 11: System architecture of *periCORE development kit* setup.

The basic components for developing customized *periCORE* applications are a PC, a *periMICA* as debugging interface and the development board itself with the *periCORE* device at its core (Figure 11).

Visual Studio Code © with the *Remote Container Extension* serves as the build and debug environment, accessing a docker container provided by *Perinet* (Figure 12).

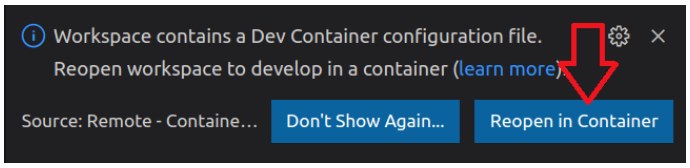


Figure 12: Using Remote Containers Extension in Visual Studio Code ©.

The docker container includes the required C++ toolchain, libraries and headers. No additional software is necessary on the Host PC except for docker and Visual Studio Code ©.

The default application workspace repository is also provided by *Perinet* and serves as the entering point to Visual Studio Code ©. Running it for the first time, the docker container will be automatically downloaded and the first build can be triggered, directly.

The firmware application is installed and debugged via a *periMICA* device that has a JLink debug device attached via USB. A *pericoredbg Container* installed on this very *periMICA* uses this interface and translates it to a TCP connection on port *5333*. In the same way the *pericoredbg Container* accesses the UART interface of the *periCORE* device providing it via TCP port *2300*. The translation of the SPI communication to the *periMICA* USB interface can thereby be done by a e.g. FTDI TTL232R-3V3 device.

Last but not least a *periSTART* device allows network communication for the *periCORE development board* for testing and debugging the HTTP REST API or MQTT Client functionality of the installed firmware.

3.2 Host PC setup

Install and prepare Visual Studio Code ©, docker and the application repository by following the guidelines from the periCORE Development Kit Setup Application Note [11].

3.2.1 Update

Software dependencies like the docker container or the application repository might be updated from time to time. To update your *periCORE Development Kit* sources, it is necessary to update the application repository:

```
cd <application-repository-basepath>/periNODE-distance
git pull
```

Listing 1: Update application repository. If the docker container has also changed it will indirectly be updated as well when using the container in Visual Studio Code ©.

This will make sure that you have the latest sources, libraries and toolchain installed.

3.3 periCORE network interface

As mentioned above, the *periCORE* device of the development board needs a network translation unit from SPE in order to communicate via Ethernet.

A *periSTART* device is used for this purpose. After connecting the development board to the *periSTART* and attaching it to a network switch, it should be available via the network. The name printed on the label (Figure 13) can be used to verify that the network functionality is working correctly.

Figure 13: periSTART device with label information.

For example using mDNS in the browser type https://<peristart-name>.local. However, refer to section Section 5 in case problems occur.

3.4 Debug interface

After installing the *pericoredbg Container* to the provided *periMICA* [11] no further configuration for the container is necessary. If accessing the *periMICA* is not working via browser due to network configuration issues again refer to Section 5.

A JLink device connects the development board to the *periMICA* via USB. The *pericoredbg Container* runs a *JLinkGDBServerCLExe* instance triggered by a socat process, providing a TCP connection via port *5333*.

As with all *Perinet Smart Components*, mDNS names can be used to access the *periMICA* containers over the network. The Visual Studio Code © debug settings therefore need to be configured with the mDNS name of the *pericoredbg Container*. The `settings.json` can be found in the application workspace repository under `.vscode`. Modify the `DEBUG_TARGET`, see also Listing 2.

```
"DEBUG_TARGET":"pericoredbg-periMICA-sernm.local:5333", // hostname of ...
```

Listing 2: Change debug target to *pericoredbg Container* mDNS name in `settings.json`.

Also in the same directory the `launch.json` needs to be adapted. Within the JSON the first object of the `inputs` array needs to be adapted, in detail the `default` value needs to be set just like for the `DEBUG_TARGET` (Listing 3):

```
"inputs": [
    {
        "id": "remote_debug",
        "description": "Enter remote debugcontainer uri",
        "default": "pericoredbg-periMICA-sernm.local:5333",
        "type": "promptString",
    }
]
```

Listing 3: Change default value to *pericoredbg Container* mDNS name in `launch.json` (Excerpt is shown).

The TCP port *5333* is pre-configured. If you need to change the TCP port or any other J-Link setting, this can be done by accessing the *pericoredbg Container* via SSH, e.g. using Putty (the mDNS name of the container can be used to reach it).

SSH access requires a one-time password, which must be generated from the *pericoredbg Container* webpage. The webpage can be accessed via the mDNS URL, or from the *periMICA* home page by clicking on the *pericoredbg Container* icon (Figure 14; refer to the periMICA User Guide [13] for more information).

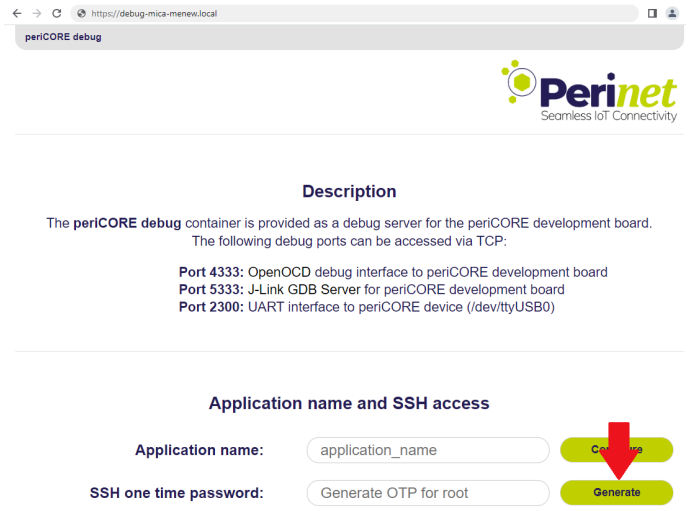


Figure 14: Excerpt from *pericoredbg Container* Web UI, using mDNS in the browser. Generate SSH One time password for user `root` access by clicking on the *Generate* button.

Note: If the access to any *pericoredbg Container* services via TCP, SSH or HTTP fails please refer to Section 5.

3.5 UART interface

As with the debug interface, the UART of the periCORE can be used for debugging via the *pericoredbg Container* on TCP port *2300*. Putty can again be used to access this interface; on

a *Linux*-based OS, telnet is often the more common tool.

After opening the TCP connection, UART messages should appear on the command terminal, provided the UART outport is configured correctly within the application firmware. (Refer to the periCORE Firmware Development Application Note [12] for more information.)

4 Software Development

4.1 periCORE Software API structure

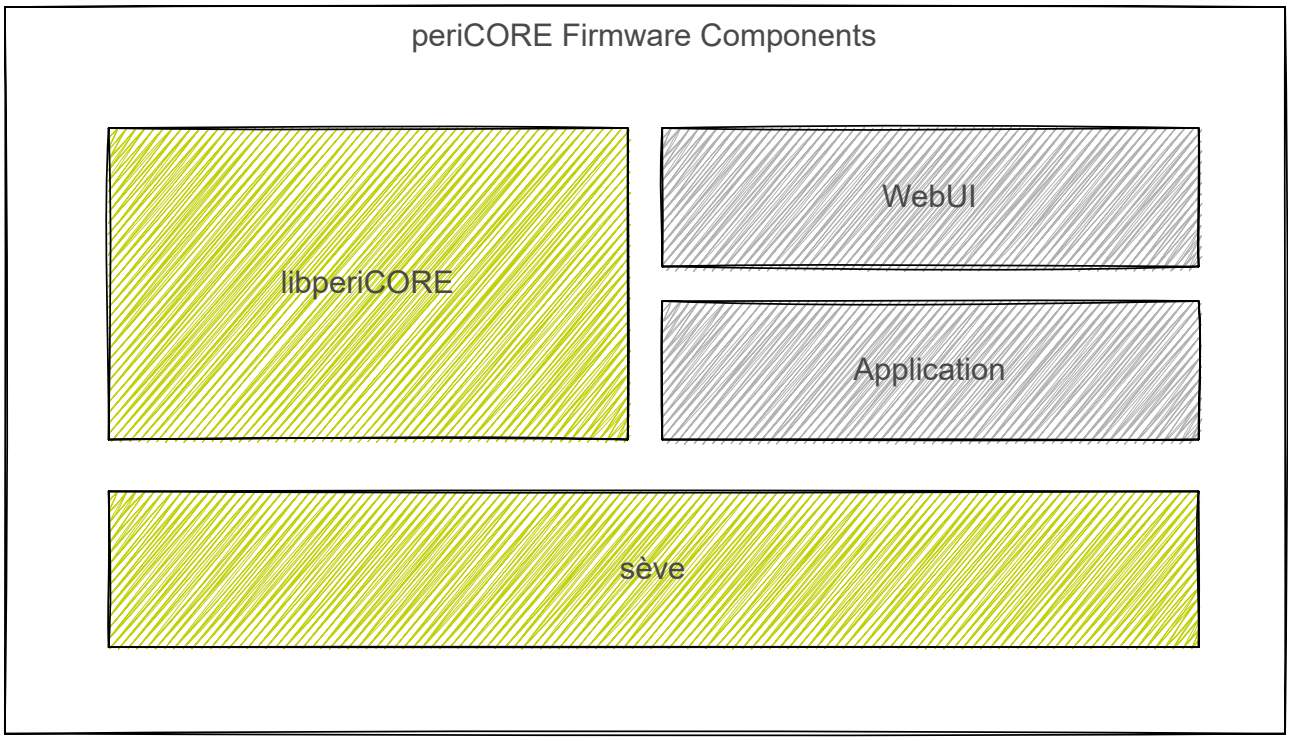


Figure 15: Firmware Architecture of *periCORE* based applications. Green boxes indicate components that are common for all applications. On the other hand, grey boxes define application specific components.

The basic idea of the *periCORE development kit* from a software perspective is to customize applications on top of the operating system (*sève*) and the *libperiCORE* library. The application part is implemented mainly by extending the functionality provided by *libperiCORE*, in particular by creating additional endpoints for both the MQTT Client and the HTTP Server.

The Web UI can also be customized on top of a minimal *Javascript* framework.

4.2 sève OS

Section 4.3 requires basic knowledge of *sève*, in particular how data is represented by the *Buffer* data type (`seve::lib::memory::Buffer`) and how outports and inports work between components, focusing on:

- `seve::eventflow::SingleValue`
- `seve::eventflow::Queue`
- `seve::eventflow::Sink`

Therefore refer to sève Operating System Datasheet [14].

4.3 Extend libperiCORE Network Services

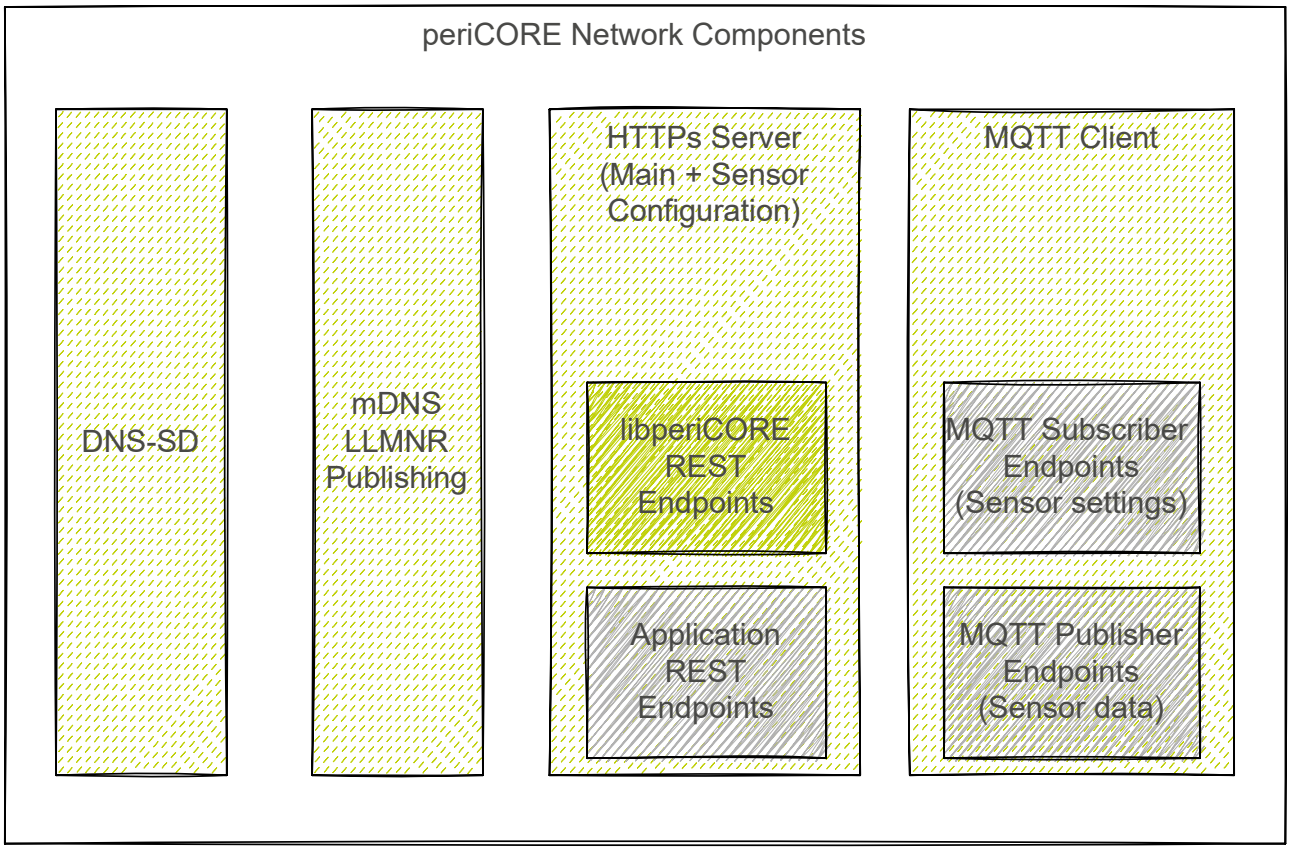


Figure 16: Network Architecture of a *periCORE* based firmware application. Green and green-dashed boxes indicate components that are provided by *libperiCORE*, hence are common for all applications. On the other hand grey boxes define application specific components, like Endpoints used for both HTTP Server and MQTT Client.

The idea of *Perinet Smart Components* is to provide sensor data and sensor interaction to the outside world. From a hardware perspective, all *Perinet Smart Components* (such as *periSTART*, *periSWITCH* and *periNODE* devices) are based on a *periCORE* module. A similar approach was followed for the software, embodied in *libperiCORE*, which provides a basic set of components, some of which can be extended.

To minimize configuration, only IPv6 link-local addresses are used for network communication across all *periCORE*-based devices. This has some implications for usage, but most distributions and browsers support IPv6 link-local addresses. If you encounter problems anyway, refer to Section 5.

The following network components provided by *libperiCORE* are used by *periCORE* based applications:

- ***HTTP Server:*** the main interface, with a REST-like API for configuring a *Perinet Smart Components* device or performing firmware updates. The HTTP interface can also be accessed via the provided Web UI, which is generic for most applications but also allows customization (see Section 4.4).

The HTTP Server backend can be modified by adding new HTTP Endpoints that perform specific actions.

As security becomes more and more important in networking, the HTTP Server provides settings for mTLS and trusted certificates, which restrict access to users with valid client certificates. See the periCORE Datasheet [10] for more information.

- ***mDNS/LLMNR Publishing:*** because IPv6 addresses can be cryptic, LLMNR and mDNS hostname publishing are used to access *Perinet Smart Components* by a unique device name, without having to type IP addresses.
- ***DNS-SD:*** DNS Service Discovery is used to browse for services of a specific type. *periCORE*-based applications use it to announce metadata about their services (such as *port*, *url* and *application* information). *periNODE* devices use it to discover the hostnames of MQTT Brokers on the network that can be used to configure the broker of the *libperiCORE* MQTT client.
- ***MQTT Client:*** a basic MQTT Client implementation is provided by *libperiCORE*, supporting TLS/mTLS features similar to the HTTP Server. Functionality is implemented by creating MQTT Client endpoints, either to publish sensor data or to subscribe to events that trigger certain processes. See Section 4.3.2 for more information.

4.3.1 Create new REST API endpoints

New REST API endpoints can be added to the *libperiCORE HTTP Server* by defining *Endpoint* (given under `libperiCORE/src/periCORE/network/http/Endpoint.h`) objects within your customized application. Upon construction of a new *Endpoint* object it registers itself on the HTTP Server (Listing 4).

```
Endpoint::Endpoint()
: seve::lib::Chainable()
, in_request(this, &Endpoint::handle_request)
, patch_out{nullptr}
, put_out{nullptr}
{
    periCORE::implementation::network::HttpServer& http_server = periCORE::
    implementation::network::HttpServer::get_object();
    periCORE::implementation::network::HttpServer*volatile test = &http_server;
    http_server.add_endpoint(this);
}
```

Listing 4: *Endpoint* constructor definition.

It provides callbacks for all HTTP methods such as *GET*, *PATCH*, *PUT*, *POST* and *DELETE*; these are called automatically by the HTTP Server when the corresponding method and URI / route is requested (Listing 5). The *Endpoint* object therefore needs a unique URI to be identified, which must be set via the `set_route` method.

```
protected:
    virtual void get(Request* r){return_status(ResponseStatus::NOT_FOUND);};
    virtual void patch(Request* r){perform_patch_post(r, patch_out);};
    virtual void post(Request* r){return_status(ResponseStatus::NOT_IMPLEMENTED);};
    virtual void put(Request* r){perform_patch_post(r, put_out);};
    virtual void del(Request* r){return_status(ResponseStatus::NOT_IMPLEMENTED);};
```

Listing 5: *Endpoint* class method declarations of HTTP callback functions.

Default implementations are already provided for the *PATCH* and *PUT* methods. Both callbacks proxy the HTTP request payload as a *Buffer* to an external component. In these cases the corresponding outports from the *Endpoint* to the receiving components need to be set (Listing 6).

```
seve::eventflow::Sink<seve::lib::memory::Buffer*>* patch_out;
seve::eventflow::Sink<seve::lib::memory::Buffer*>* put_out;
```

Listing 6: *Endpoint* outport declarations for *PATCH* and *PUT* methods.

However, all HTTP method callbacks are virtual and can be overridden by deriving a new class from the base class *Endpoint*. The *HTTPEndpoint* defined in `libperiCORE/src/periCORE/network/http/HTTPEndpoint.h` is an example of such a derived class (Listing 7).

```
class HTTPEndpoint : public periCORE::network::http::Endpoint
{
public:
    seve::eventflow::Queue<HTTPEndpoint, seve::lib::memory::Buffer> inPort_buffer{this,
    &HTTPEndpoint::update};

private:
    void get(periCORE::network::http::Request* request) override;
    void update(seve::lib::memory::Buffer *buffer);
    seve::lib::memory::Buffer *cached{nullptr};
};
```

Listing 7: Declaration of *HTTPEndpoint* class derived from *Endpoint*.

It caches the latest *Buffer* (typically a JSON message) provided by an external component and sends it as the response to the HTTP client when an HTTP *GET* request is received on the corresponding URI (see also Figure 17).

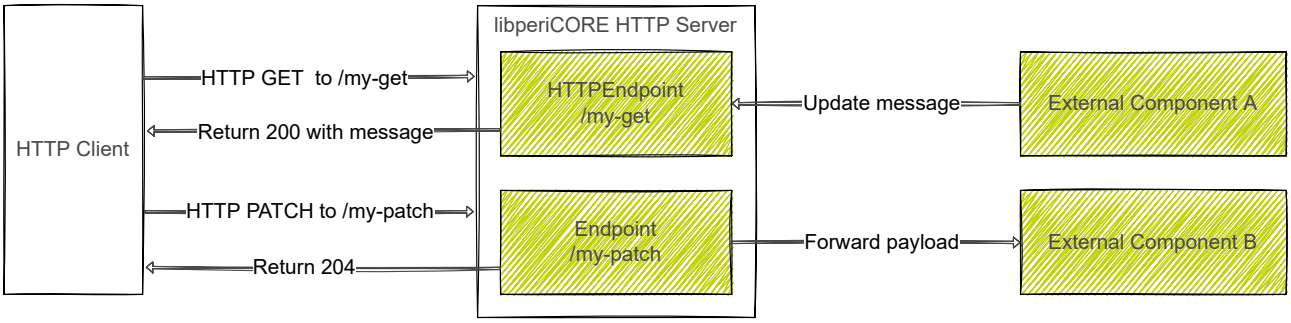


Figure 17: Example application HTTP Server with two endpoints, one for either HTTP GET and PATCH request handling.

JSON is used as the data representation format for HTTP request and response communication.

When mTLS is active in the application, the `handle_request` method of the *Endpoint* class determines which user role is required for the corresponding URI and HTTP method combination. The following user roles are implemented by default (Listing 8):

- *PATCH* / *PUT* requests require the *ADMIN* role; URIs starting with */config* or */sample* require *SUPER* rights.
- *GET* / *DELETE* / *POST* requests: the default user role is *READER*. This is intended for *GET* requests, but is also applied to *DELETE* and *POST* since these HTTP methods are not yet used or implemented.

```
if(!request->method.compare("PATCH"))
{
    if( route.compare("/sample") == 0 ||
        route.compare("/sample/gpio1") == 0 ||
        route.compare("/sample/gpio2") == 0 ||
        route.compare("/config") == 0 ||
        route.compare("/config/reset") == 0)
    {
        required_role = periCORE::security::UserRole::SUPER;
    }
    else
    {
        required_role = periCORE::security::UserRole::ADMIN;
    }
}
else if(!request->method.compare("PUT"))
{
    required_role = periCORE::security::UserRole::ADMIN;
}
```

Listing 8: Excerpt from handle_request method of *Endpoint* class.

The three user-role types for *periCORE*-based applications are defined as follows:

- ***READER***: as the name suggests, intended to allow users only to read specific settings or information.
- ***SUPER***: additionally allowed to change sensor configuration and sensor settings.
- ***ADMIN***: has full access; allowed to perform firmware updates and apply main configurations, such as security settings.

Also see periCORE Firmware Development Application Note [12] for a complete example on how to use customized HTTP endpoints within your application.

4.3.2 Create new MQTT Client endpoints

Similar to the *libperiCORE HTTP Server*, the *libperiCORE MQTT Client* handles MQTT endpoints either by publishing a message when an *MQTTPublisherEndpoint* triggers its outport (e.g. by a timing event), or by forwarding a received MQTT message to the *MQTTSubscriberEndpoint* that subscribes to the corresponding MQTT topic.

Both MQTT endpoint classes must be defined with an MQTT topic on construction of a new object (Listing 9).

```
class MQTTEndpoint: public seve::lib::Chainable
{
public:
    MQTTEndpoint(const MQTTTopic&& topic);
    const std::string_view get_topic(){return mqtt_topic.get();}
protected:
    const MQTTTopic mqtt_topic;
};
```

Listing 9: *MQTTEndpoint* class declaration serving as base class for both *MQTTPublisherEndpoint* and *MQTTSubscriberEndpoint* classes.

Also on construction of an *MQTTSubscriberEndpoint* object it automatically registers itself on the *libperiCORE MQTT Client* (Listing 10):

```
MQTTSubscriberEndpoint::MQTTSubscriberEndpoint(const MQTTTopic&& topic)
    : MQTTEndpoint{std::move(topic)}
{
    periCORE::implementation::network::MQTTClient& mqtt_client = periCORE::
    implementation::network::MQTTClient::getInstance();
    mqtt_client.register_subscriber_endpoint(*this);
}
```

Listing 10: *MQTTSubscriberEndpoint* constructor definition.

The *MQTTPublisherEndpoint* is not added directly to the *libperiCORE MQTT Client*; instead, the outport of the endpoint is connected to the publish inport of the *libperiCORE MQTT Client* (Listing 10). The endpoint thus indirectly triggers publishing when a new message is received from the connected external component (see Figure 18).

```
using namespace periCORE::implementation::network;

MQTTPublisherEndpoint::MQTTPublisherEndpoint(const MQTTTopic&& topic)
: MQTTEndpoint{std::move(topic)}
, outport_mqtt{*MQTTClient::getInstance().get_inport_publish()}
{
}
```

Listing 11: *MQTTPublisherEndpoint* constructor definition.

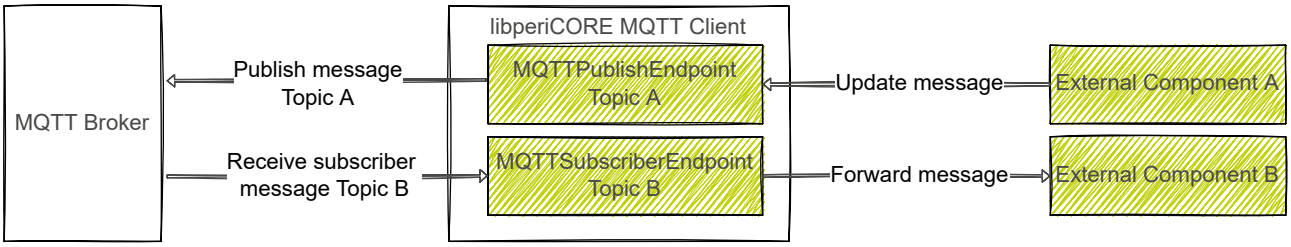


Figure 18: Example application MQTT Client with one *MQTTPublisherEndpoint* and one *MQTTSubscriberEndpoint*.

- *MQTTPublisherEndpoint* (`libperiCORE/src/network/MQTTPublisherEndpoint.h`):

  Similar to the *HTTPEndpoint* the *MQTTPublisherEndpoint* class expects a *Buffer* handler from an external component publishing this latest update upon reception. But in contrast to the *HTTPEndpoint* the *Buffer* is not cached in the endpoint, instead it is published directly. The connection of the *MQTTPublisherEndpoint* inport to the external component is done by assigning the `get_inport` method.

- *MQTTSubscriberEndpoint* (`libperiCORE/src/network/MQTTSubscriberEndpoint.h`):

  Basically serves as a proxy to transfer a *Buffer* to an external component upon reception of a MQTT message with the according topic. The connection to the receiving component is done by calling the `set_outport` method of the *MQTTSubscriberEndpoint* right after construction.

As with the HTTP Server, JSON is used as the data representation format for MQTT data communication.

Also see periCORE Firmware Development Application Note [12] for a complete example on how to use customized MQTT Client endpoints within your application.

4.4 Web User Interface

4.4.1 Sources and image creation

Frontend source files of *periCORE* applications are compiled into a separate Web UI binary, which is bundled together with the application firmware into a firmware update image. The Web UI cannot be installed or updated via the debugging tools; instead, the firmware update function via the Web UI or REST API call of the *periCORE* device must be used.

The sources of the Web UI binary consist mainly of js and html files. Most of the functionality is implemented in *Javascript* to give a web-application-like feel: for example, only the specific page content is loaded when navigating between Web UI tabs. The header, footer and the basic css and js are loaded only once; only the actual content in between is reloaded on demand. This also keeps the request load low on the *periCORE* device, which has limited performance, especially for HTTP with TLS features.

Also for performance reasons, the Web UI Makefile encodes css and png file links as Base64 strings and substitutes them into the corresponding Javascript and html files. png is the recommended image format, since the total Web UI binary image size is limited to *256 KB*.

Again see periCORE Firmware Development Application Note [12] for an in depth example on how to develop customized Web applications for *periCORE* devices.

4.4.2 TLS session resumption

Another way of improving Web UI performance is to make use of TLS session resumption. A typical HTTP call to a *periCORE*-based device with TLS enabled takes about *700 ms*; with mTLS enabled it takes even longer, about *1300 ms* per request. With TLS session resumption, however, the TLS handshake is performed only on the first request; subsequent requests reuse the initial TLS session on both client and server, so they take only *100 ms* to *200 ms* each, regardless of whether mTLS is used.

The *periCORE* HTTP Server backend supports TLS session resumption; however, some browsers have it disabled on the client side. Newer Firefox versions appear not to use it any longer, while Microsoft Edge and Google Chrome currently still support it.

4.4.3 Html Dynamic Content Modification

As mentioned above, the content of the Web UI can be divided into header, content and footer. The header and footer remain constant; only the content in the center is dynamically changed (see Figure 19).

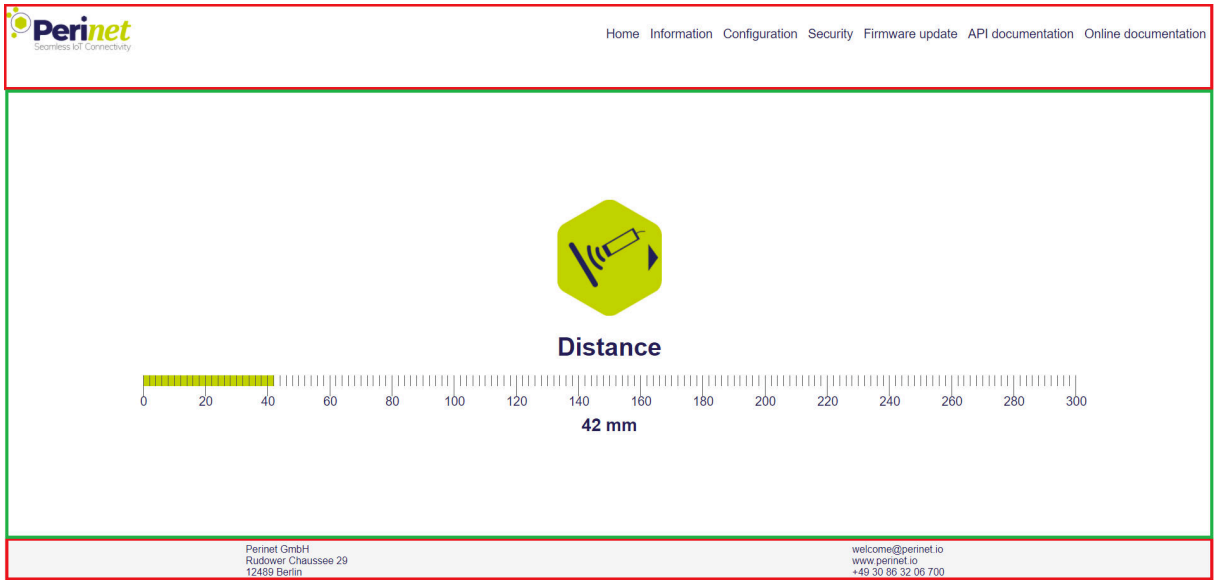


Figure 19: Web UI structure of *periCORE* based applications. Red boxes indicate static content (header and footer), whereas the actual content (green box) is dynamically modified when navigating to other web pages using the header tabs.

To allow content to be loaded automatically, no html files need to be modified; instead, html DOM elements are created by Javascript functions. For example, the content function that creates the DOM elements for the Firmware Update web page (Listing 12, taken from the application repository path `webui/fs_root/js/update.js`):

```
function dom_update() {
    return `<div id="overlay"></div><div>
    <figure>
        <div class="update-container">
            <img src="images/update-plain.png" alt="periNODE" id="update-plain">
            <img src="images/update-arrows.png" id="update-overlay">
        </div>
    </figure>

    <h1 id="update-header">Firmware update</h1>
    </div>
    <div>
    <div><input type="file" name="fileUpload" id="fileUpload"></div>
    <div><input style="margin-left:2pt" type="submit" id="uploadbutton" alt="Update
    Firmware" value="Upload file"></div>
    <div id="upload_label" for="upload_status">
        <div class="loader">
            <div class="check">
                <!-- <span class="check-one"></span>
                <span class="check-two"></span> -->
            </div>
        </div>
    </div>
    </div>`;
}
```

Listing 12: html content created for Firmware Update Web page using Javascript.

The DOM elements in the code can directly be accessed from within Javascript. Looking at the example case defining the `dom_update` function is not enough. It needs to be registered in the framework (Listing 13 taken from `webui/fs_root/js/update.js`).

```
function reload_update() {
    $("content_id").innerHTML = dom_update();
    document.title = "periNODE Web Server | Firmware update";
    $('uploadbutton').addEventListener('click', onSelectFile, false);
}
update_content("update", reload_update);
```

Listing 13: Registering DOM content creation in Javascript Framework.

The `update_content` function therefore needs to be called by passing a callback function (`reload_update` in this case), which in turn is called when the user enters the corresponding web page. The first parameter of `update_content` is an identifier for the web page, which is used by the framework (`peri_base.js`) to identify the callback (see also the calls to the `reload` functions in Listing 14).

4.4.4 Html Static Content Modification

If changes to header and footer need to be made this can be done by modifying the according `dom_footer` / `dom_header` functions in the `webui/fs_root/js/peri_base.js` file from the application repository folder. (See example for the header function in Listing 14)

```
function dom_header(){
    return `<header class="header">
        <a href="https://www.perinet.io" target="_blank" class="logo"> <img src="images/
    perinet-logo.png" alt="Perinet Logo" width="180"></a>
        <input class="menu-btn" type="checkbox" id="menu-btn" />
        <label class="menu-icon" for="menu-btn"><span class="nav-icon"></span></label>
        <ul class="menu">
            <li><a href="home.html" onclick="return reload('home')">Home</a></li>
            <li><a href="info.html" onclick="return reload('info')">Information</a></li>
            <li><a href="config.html" onclick="return reload('config')">Configuration</a
    ></li>
            <li><a href="security.html" onclick="return reload('security')">Security</a
    ></li>
            <li><a href="update.html" onclick="return reload('update')">Firmware update
    </a></li>
            <li><a href="doc/api.proto" target="_blank">API documentation</a></li>
            <li><a href="https://docs.perinet.io" target="_blank">Online documentation</
    a></li>
        </ul>
    </header>`;
}
```

Listing 14: Javascript function to change header DOM content.

4.4.5 Create new Web page

To add a new page to the Web UI, follow these steps:

- Copy the `home.html` from `webui/fs_root` folder into the same destination and name it accordingly, e.g. `custom.html`.
- In the `webui/fs_root/js/peri_base.js` file adapt the `dom_header` function by adding a new navigation link (example Listing 15) to the desired position.

```
<li><a href="custom.html" onclick="return reload('custom')">Mytab</a></li>
```

Listing 15: Navigation link for new custom web page within peri_base.js file.

- Now create a new js file under `webui/fs_root/js_node`, e.g. named `custom.js`. Here the callback needs to be defined when the user enters the new web page (simple example in Listing 16).

```
function reload_custom() {
    $("content_id").innerHTML = "<span>My custom content</span>";
    document.title = "periNODE Web Server | Custom";
}
update_content("custom", reload_custom);
```

Listing 16: Example Javascript file for new page.

The `update_content` call is needed once, in case the js file is loaded for the first time.

- Now the `reload_custom` callback needs to be registered, i.e. added in the `peri_base.js` file as follows:

```
function load_custom() {
    if(!load_js_script("js_node/custom.js")) {
        update_content("custom", reload_custom);
    }
}
```

Listing 17: Register callback function for new web page in peri_base.js file.

The function must be named `load_<uri>`. The URI must match the value passed to the `reload` function for the navigation link (see again Listing 15). The `update_content` function expects the URI as its first parameter and the callback defined in the corresponding js file as its second parameter, exactly as it is called in the js file.

Note: You might wonder why the `update_content` function is only loaded if the Javascript file was already loaded. This is because the Javascript file with the actual reload callback is loaded asynchronously: in the example of the `load_custom` function calling `load_js_script`, there is no guarantee that the Javascript file is loaded by the time `load_js_script` returns. The loaded Javascript file therefore calls `update_content` once itself.

4.4.6 Loading css files

Additional stylesheet files can be added by calling the corresponding Javascript function `load_css_script` (see Listing 18, taken from `webui/fs_root/js_node/home.js`).

```
load_css_script("css_node/ruler.css");
```

Listing 18: Example call for loading css file.

Note: When modification or replacement of *Perinet* default style sheets is necessary, modify the `set_style` function within `webui/fs_root/js/peri_base.js` by loading your customized style sheets instead.

It is important to understand that the Web UI Makefile replaces the css links in all js and html files with their Base64 string representation (compare Listing 18 and Listing 19). This means that css files do not require a separate HTTP request, which keeps the load low on the *periCORE* application HTTP server.

```
load_css_script("data:text/css;base64,LnJ1bGVyIHsgbWFyZ2luOiBhdXRvOyB...");
```

Listing 19: Example call for loading css file after building. Out of convenience reasons the Base64 string is not shown entirely.

Note: The Makefile also does not add these css files to the Web UI binary, hence they are not available via separate HTTP request, as well. This is intended to keep the Web UI binary size low, which allows only a total size of 256 KB.

4.4.7 Loading image files

As with css files, png image file links are replaced by the Web UI Makefile in all js and html files with their Base64 string representation, in order to keep the load low on the *periCORE* application HTTP server.

Note: As with css files, the Makefile does not add the actual image files to the Web UI binary. To support other file formats such as jpeg, we recommend using the Base64 string representation as well; this would require changes to the Web UI Makefile.

4.4.8 Debugging Web UI changes

To test a newly implemented Web UI, you need to create the firmware update image and update the *periCORE* from the development board via the REST API or the Web UI. The debugger via the *pericoredbg Container* cannot be used here, because the Web UI files are not reloaded into the flash memory. See also the periCORE Firmware Development Application Note [12] for more information.

Note: The IPv6 link-local address and hostname for the *periCORE* application services can be set within the application repository under `src/defaultNodeInfo.cc`. The IPv6 address is derived from the MAC address as `fe80::<mac-address-formatted>:0`.

5 Troubleshooting

I cannot reach the webpage of the *periMICA* device.

Make sure that your OS supports mDNS resolution and IPv6 link-local addresses. Although the *periMICA* device also supports IPv4, the *periMICA* containers and *Perinet Smart Components* can only be accessed using IPv6 link-local addresses. In a command terminal (under *Windows* press the **Windows** key and type `cmd`), check that `ping` works correctly (Listing 20).

```
ping -6 <peri-device-name>.local
```

Listing 20: Ping command to *Perinet* device.

If using versions below *Windows 10* you might need to install some mDNS client software, like Bonjour.

I cannot access IPv6 link-local addresses with certain browsers

This mainly affects you if your host OS is Debian- or Ubuntu-based, because most browsers struggle with IPv6 link-local address zone identifiers on those operating systems. Firefox can be an exception here, but it is not recommended because the firmware update from Firefox does not work for *Perinet Smart Components*. There are two options:

- (Recommended) using the *periMICA* as proxy device via *Remote Devices* feature by forwarding HTTP requests to *periMICA* containers and *Perinet Smart Components*. See periMICA User Guide [13] for more information on this.
- using socat to proxy TCP connections from IPv6 link local to the IPv4 address of your local host for HTTP services (Listing 21)

```
socat TCP4-LISTEN:<some-custom-port>,fork,reuseaddr \
    TCP6:[<peri-device-llv6>%<localhost-nic>]:443&
```

Listing 21: General command to proxy HTTP connections.

You can get the IPv6 link local address of your *periMICA* container or *Perinet Smart Components* device by using avahi or a simple `ping` command to the mDNS name of the device (Listing 20). For example if you wanted to forward HTTP connections from a *periCORE* with the mDNS name `periCORE-n8ipe` that has the IPv6 address `fe80::742e:dbcf:21a4:0` and your localhost network interface was `eth0` the socat process would be like in Listing 22.

```
socat TCP4-LISTEN:8100,fork,reuseaddr \
    TCP6:[fe80::742e:dbcf:21a4:0%eth0]:443&
```

Listing 22: Proxy command example.

Now in the browser simply type https://localhost:8100/ and you should be able to see the Web UI of your *periCORE* device.

6 Labeling and Ordering

Labeling

Unlike the periCORE module, the periDEVboard 3.3 itself carries no labeling. For an interpretation of the periCORE module label, please refer to the periCORE datasheet.

Ordering

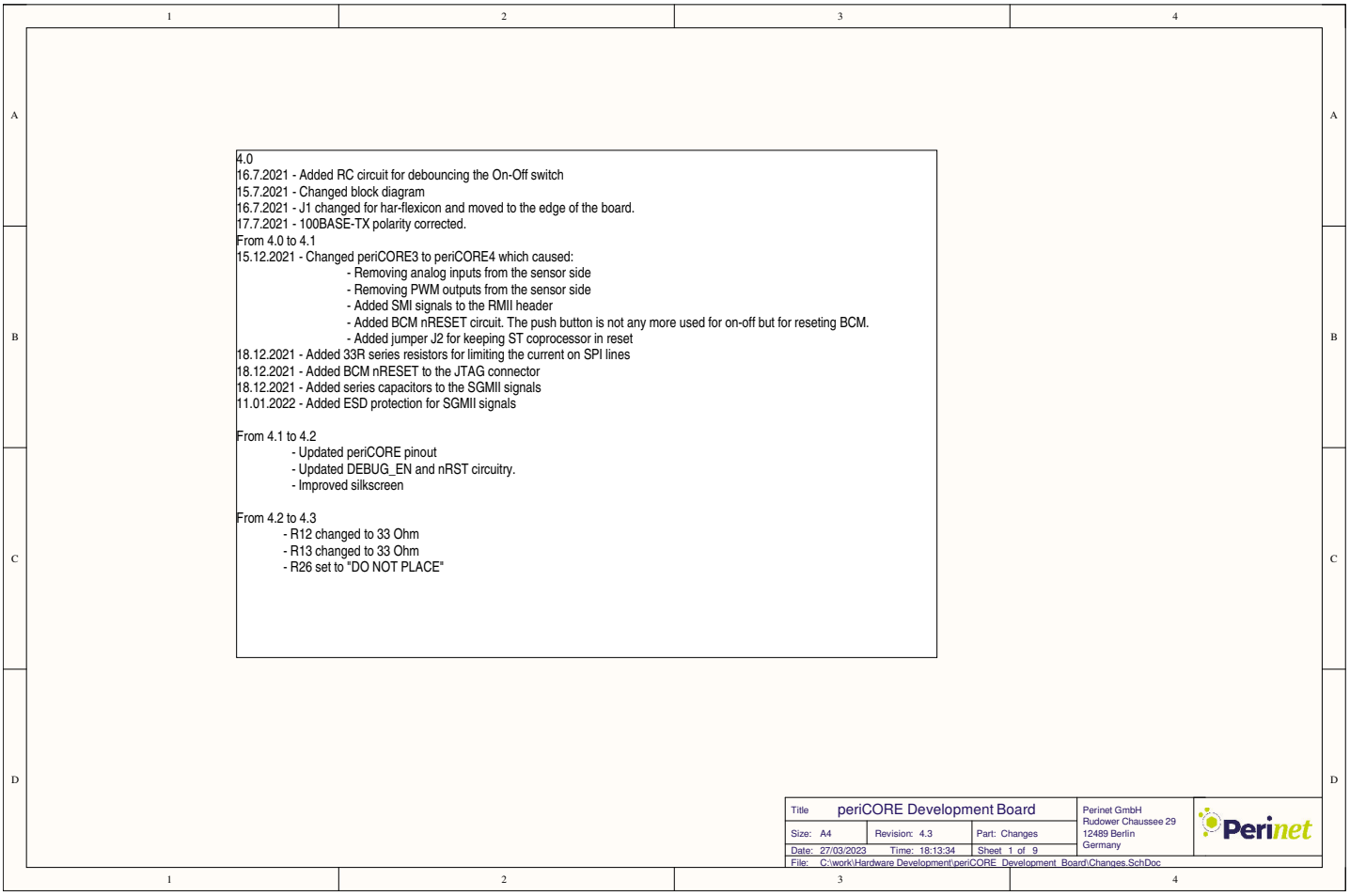
To order the periDEVboard 3.3, please contact *sales@perinet.io*.

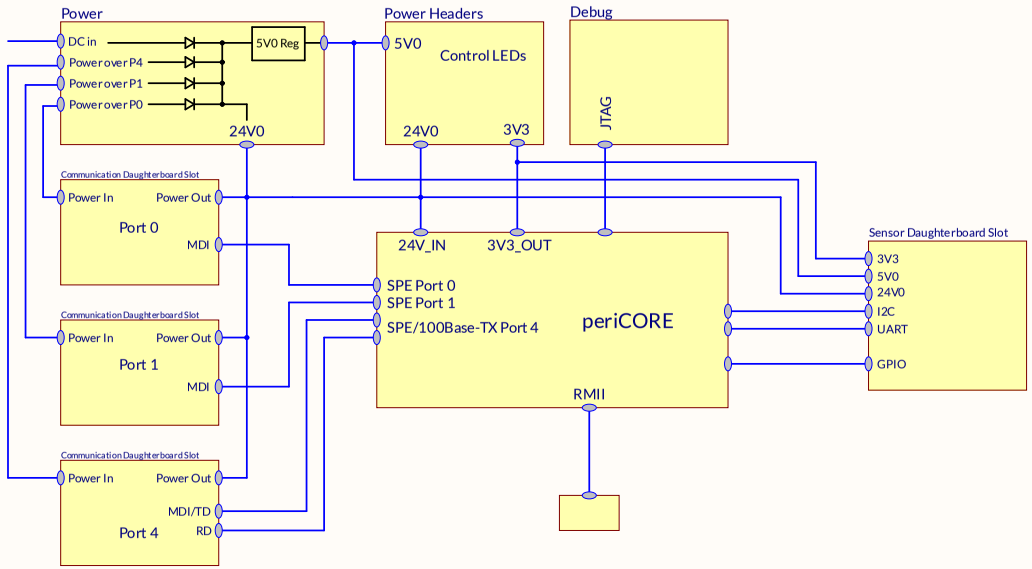
7 Contact & Support

For customer support, please call us at **+49 30 863 206 701** or send an e-mail to *support@perinet.io*.

For complete contact information visit us at www.perinet.io

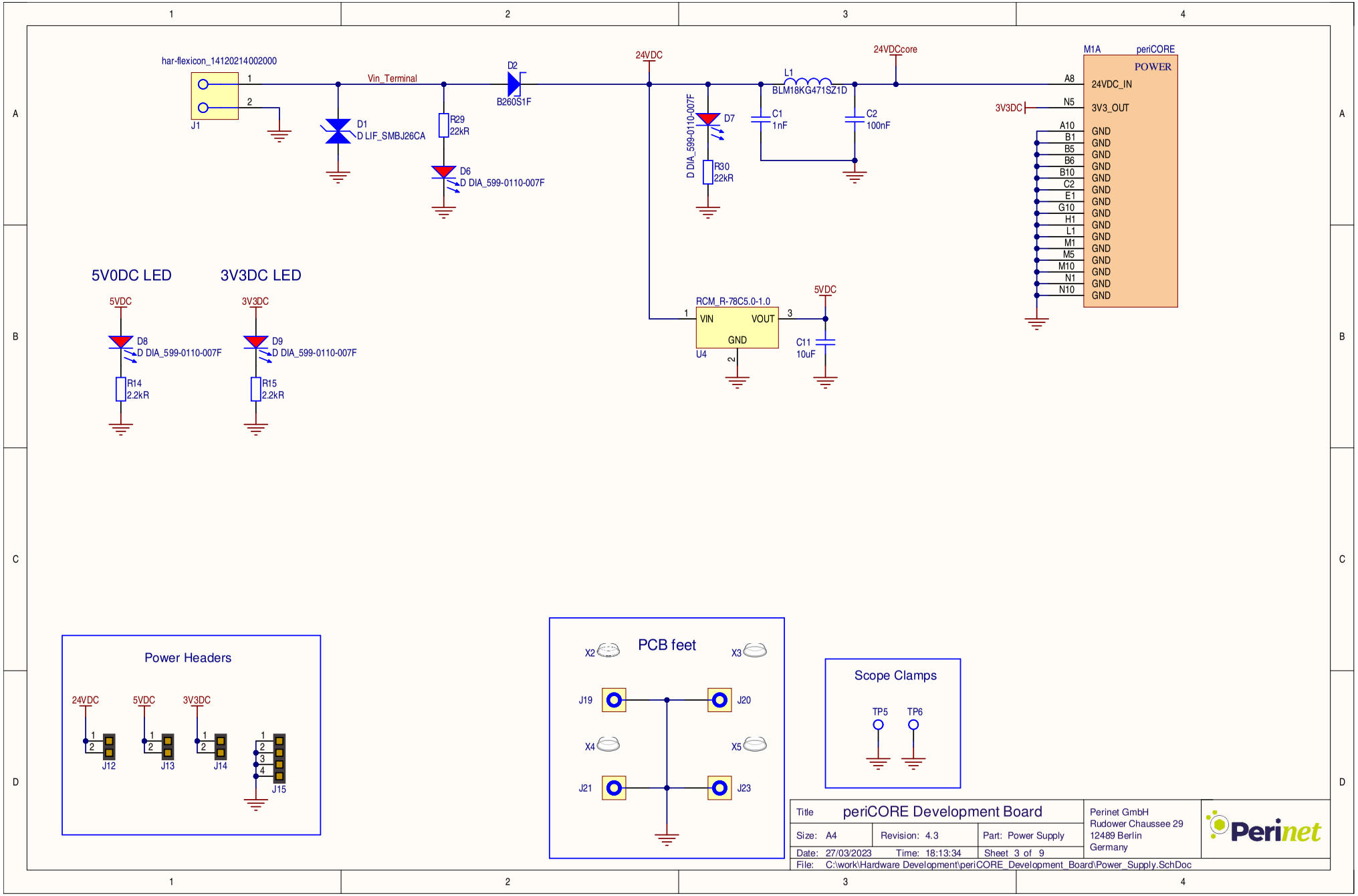
A Development Board



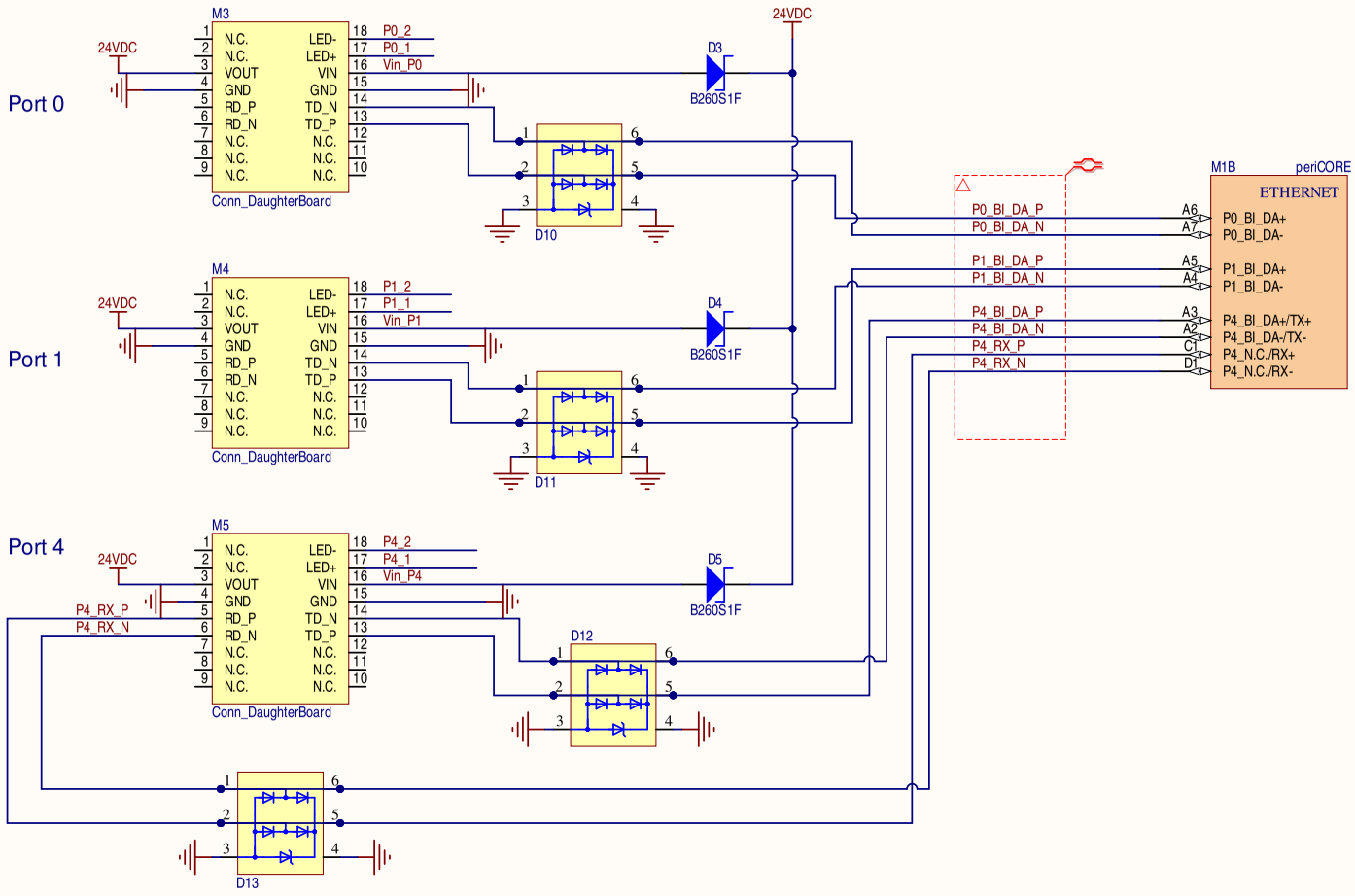


Logo Perinet Logo Perinet

Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: Overview	
Date: 27/03/2023	Time: 18:13:34	Sheet 2 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\Overview.SchDoc			

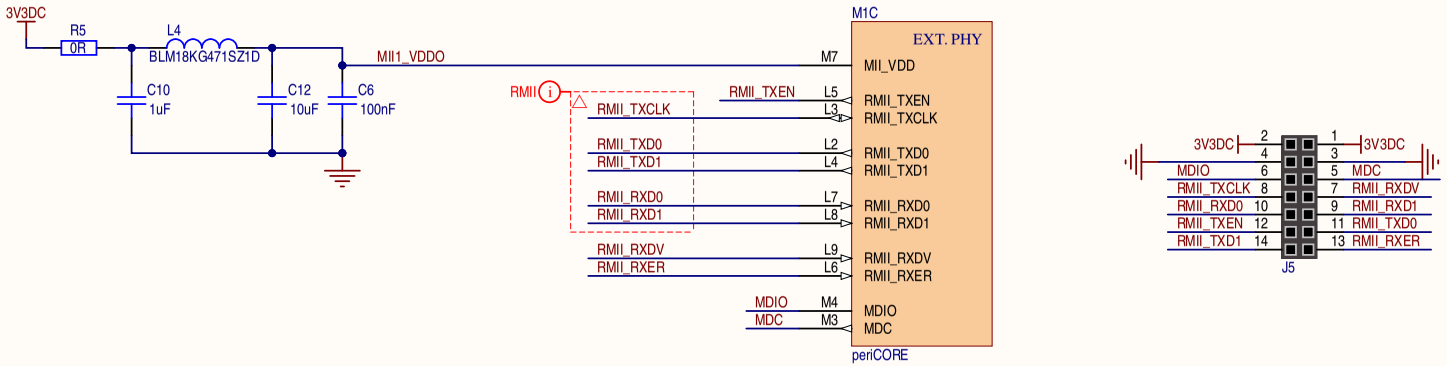


Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: Power Supply	
Date: 27/03/2023	Time: 18:13:34	Sheet 3 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\Power_Supply.SchDoc			

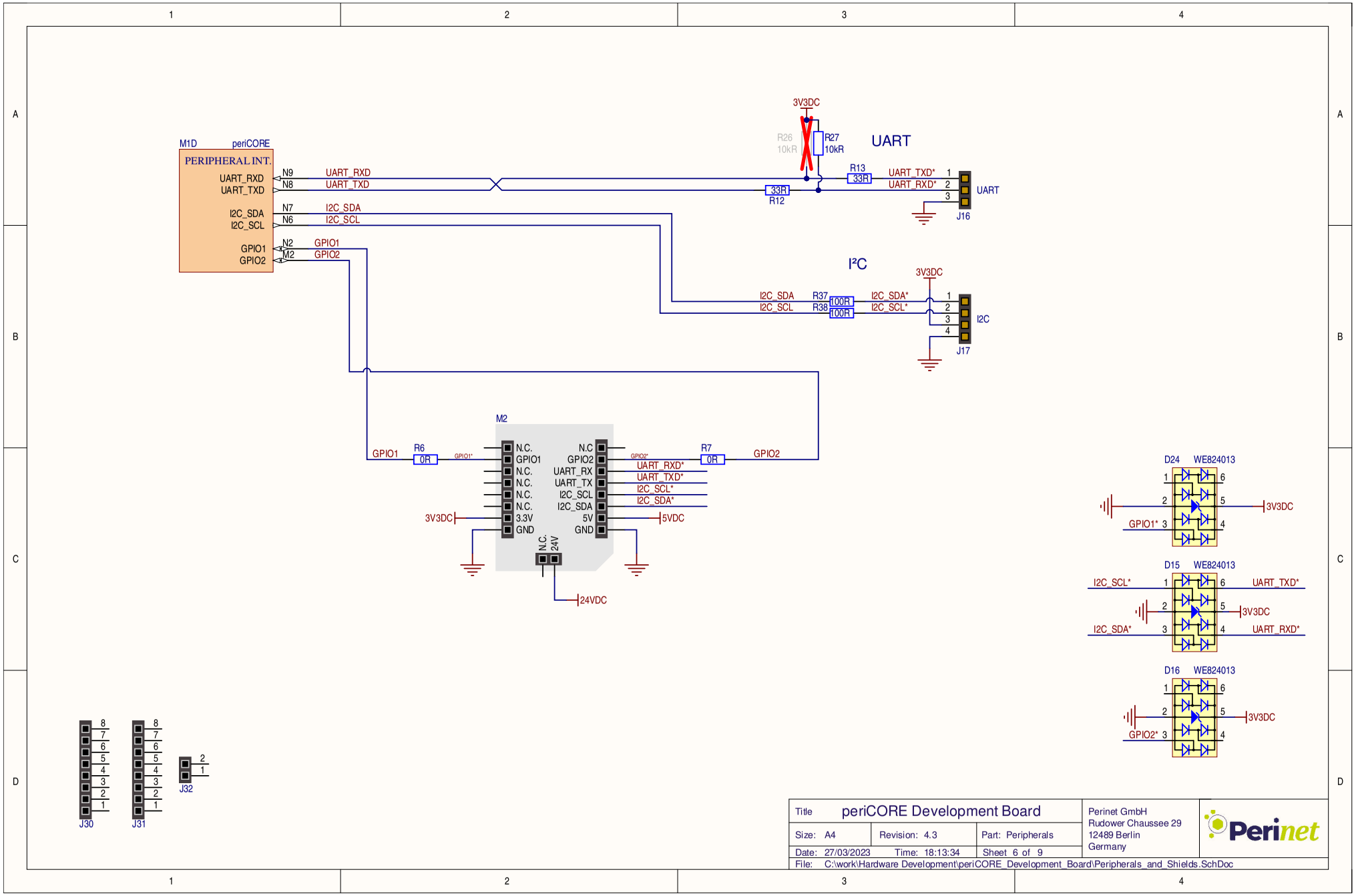


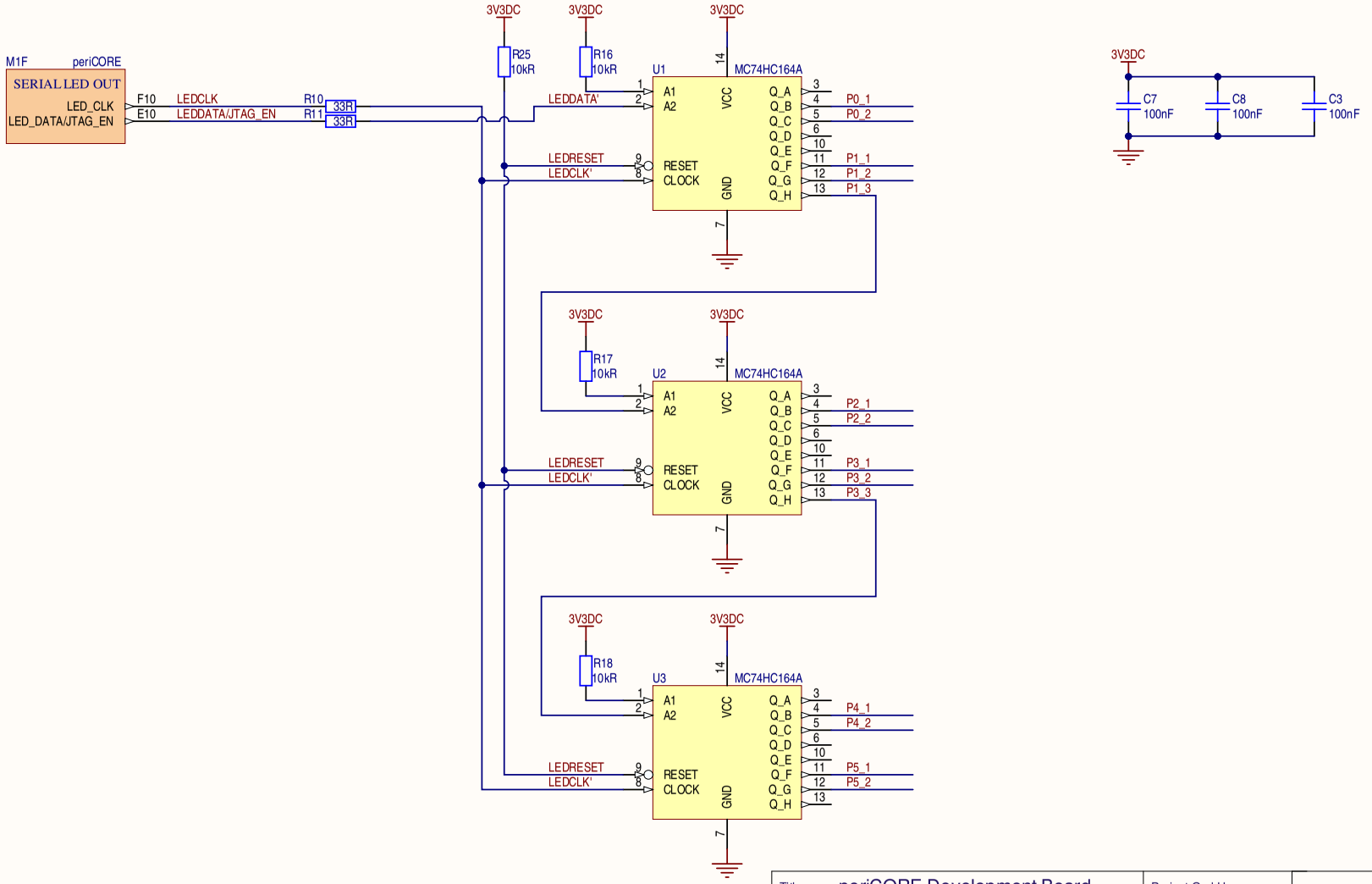
J28 J29 J26 J27 J24 J25

Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: SPE & 100BASE-TX	
Date: 27/03/2023	Time: 18:13:34	Sheet 4 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\Network_Interfaces.SchDoc			

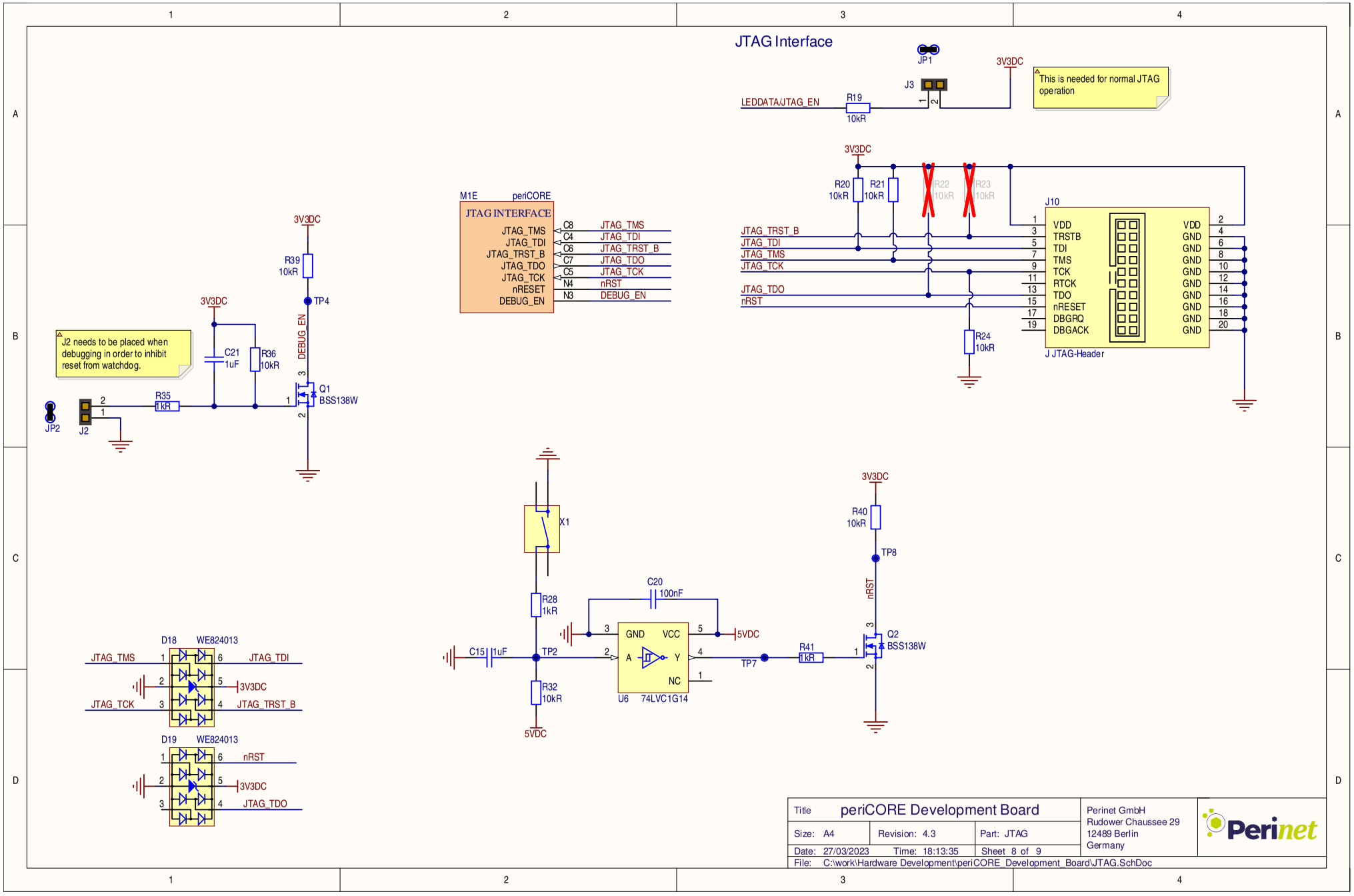


Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: RMII	
Date: 27/03/2023	Time: 18:13:34	Sheet 5 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\RMII.SchDoc			


M1D periCORE
PERIPHERAL INT.
UART_RXD
UART_TXD
I2C_SDA
I2C_SCL
GPIO1
GPIO2
N9 UART_RXD
N8 UART_TXD
N7 I2C_SDA
N6 I2C_SCL
N2 GPIO1
M2 GPIO2
3V3DC
R26 10kR
R27 10kR
UART
R13 33R
R12 33R
UART_TXD*
UART_RXD*
J16
I²C
R37 100R
R38 100R
I2C_SDA*
I2C_SCL*
J17
M2
GPIO1
R6 0R
R7 0R
GPIO2
N.C.
3.3V
GND
5V
UART_RX
UART_TX
24V
5VDC
24VDC
D24 WE824013
D15 WE824013
D16 WE824013
J30
J31
J32
Title periCORE Development Board
Size: A4
Revision: 4.3
Part: Peripherals
Date: 27/03/2023
Time: 18:13:34
Sheet 6 of 9
File: C:\work\Hardware Development\periCORE_Development_Board\Peripherals_and_Shields.SchDoc
Perinet GmbH
Rudower Chaussee 29
12489 Berlin
Germany
Perinet



Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: Link Status LEDs	
Date: 27/03/2023	Time: 18:13:35	Sheet 7 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\Link Status LEDs.SchDoc			



Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.3	Part: JTAG	
Date: 27/03/2023	Time: 18:13:35	Sheet 8 of 9	
File: C:\work\Hardware Development\periCORE_Development_Board\JTAG.SchDoc			

B Network daughterboards

The network communication daughterboards are carrier boards for different connectors. They have two rows of 9-pin male headers, and are attached into the female header counterpart receptacles on the development board. A drawing of a typical network daughterboard is shown in Figure 20.

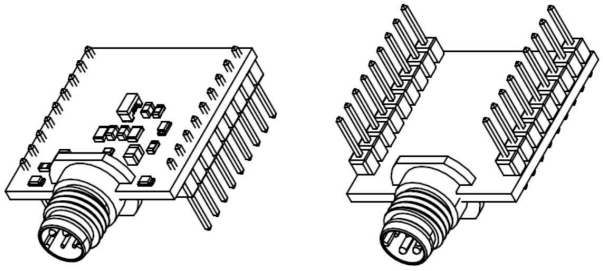


Figure 20: M8 Hybrid Male daughterboard drawing.

The pinout of the network communication daughterboard is shown below in Figure 21.

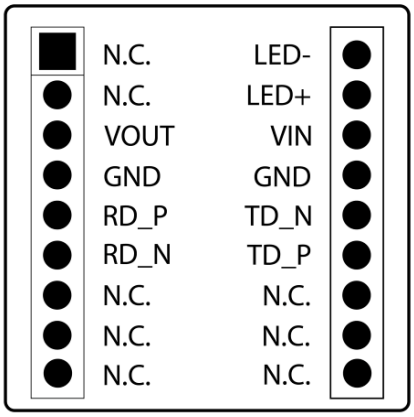


Figure 21: Network communication daughterboard pinout.

The daughterboard interface uses the following signals:

- Differential pair TD (`TD_N` and `TD_P`) for data transmission/reception in 100BASE-T1. If a daughterboard is used for 100BASE-TX, which can be the case for port 4, this differential pair is used only for data transmission from the periCORE.
- Differential pair RD (`RD_N` and `RD_P`) is used for data reception in 100BASE-TX.
- `LED+` and `LED-` are PHY signals used to control the link indicator LEDs.
- `VIN` is the positive terminal of the external power supply that can power the development board via the connector (e.g. a daughterboard). This terminal is used in hybrid cabling to supply power to the development board via a hybrid cable.

- `VOUT` is a positive power supply terminal driven from the development board to the daughterboard connector. On the development board, this signal is directly connected to the main power supply for the *periCORE*. In hybrid cabling, this signal is used to supply power to the nodes on the other side of the hybrid cable.
- `N.C.` means not connected. The function of these pins is not defined at the moment; they are reserved for future use.

The difference between the daughterboards for 100BASE-TX and 100BASE-T1 is that the former uses both the TD and RD differential pairs, while the latter uses only TD.

The daughterboards currently available cover the following connectors:

- For 100BASE-T1 networks:
  - M8 male, style 6P-M8C (without shield) defined in IEC 63171-6:2021 [6]
  - M8 female, style 6J-M8C (without shield) defined in IEC 63171-6:2021 [6]
  - HARTING T1 Industrial Jack, style 2J-L defined in IEC 63171-6:2021 [6]
  - METZ CONNECT spring clamps, for open-end cables
- For 100BASE-TX networks:
  - HARTING ix
  - RJ45

The following sections provide more details and schematics for the aforementioned network daughterboards.

B.1 M8H Male DaughterBoard

The M8H Male DaughterBoard is a network daughterboard which allows using the M8 male connector according to IEC 63171-6:2021 (style 6P-M8C without shielding) [6] with the *periCORE Development Board*. The daughterboard is shown in Figure 22. The connector is intended for hybrid SPE cables with two wire pairs, where one pair is used for the power supply, and the second pair is a differential 100BASE-T1 pair (SPE). The face of the connector with the signal definitions is given in Figure 23.

Figure 22: M8H Male DaughterBoard

Contact	PMA signal	Wire colour
1	BI_DA+	Blue
2	BI_DA-	White
3	U	Red
4	GND	Black

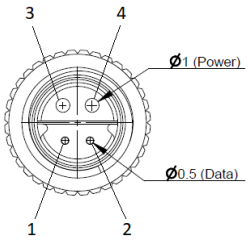


Figure 23: M8H Male connector drawing and pinout [16]

The pinout of the M8H Male DaughterBoard is shown in Figure 24.

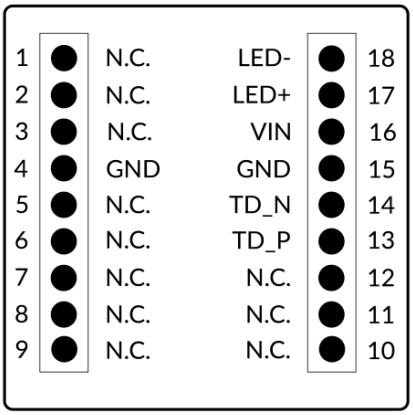
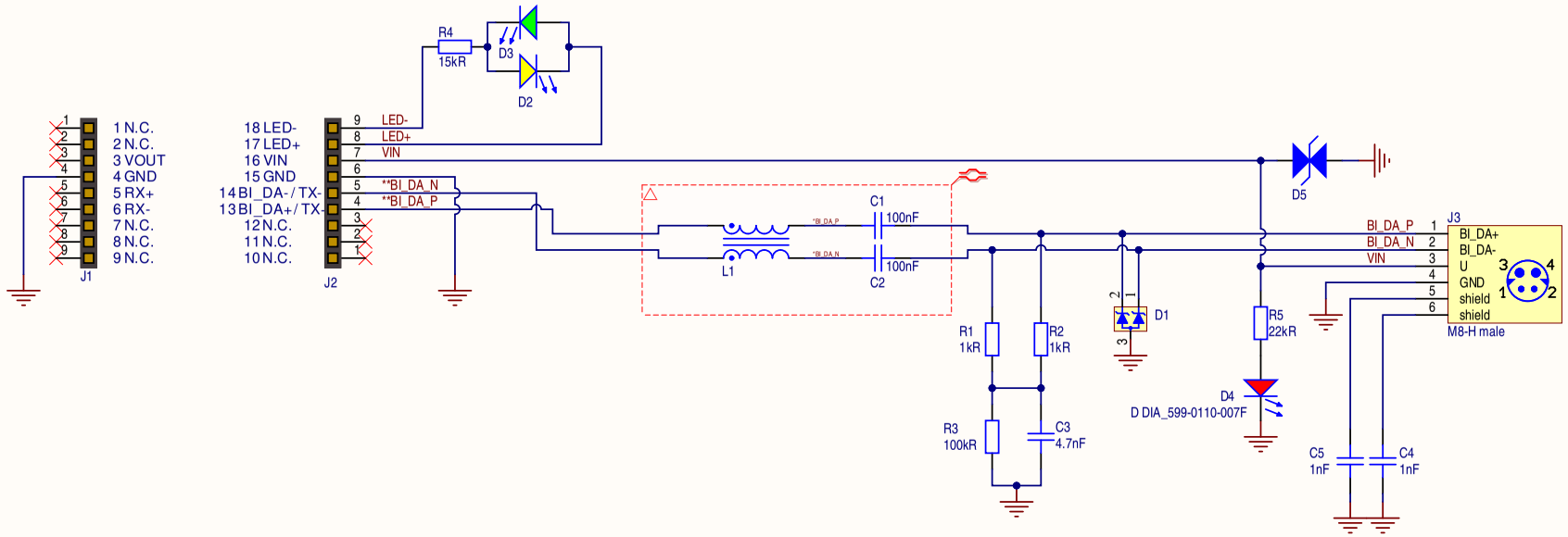


Figure 24: M8H Male daughterboard pinout

The schematic of the M8H Male DaughterBoard is given on the next page:

- Pin 3 of the M8H Male connector is connected with the pin 16 of the daughterboard which means that the development board can be supplied with power over this daughterboard.
- There are two LEDs (D2 and D3) for showing various link status indicators. The indicators can be configured in software.
- M8H Male daughterboard is intended to be used on the 100BASE-T1 port which is configured as slave.



LOGO1 LOGO2

Title	M8H Male DaughterBoard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: Overview	
Date: 05/12/2023	Time: 15:09:37	Sheet 1 of 1	
File: C:\work\Hardware Development\M8HM_Daughterboard\M8HM_Daughterboard.SchDoc			

B.2 M8H Female DaughterBoard

The M8H Female DaughterBoard is a network daughterboard which allows using the M8 female connector according to IEC 63171-6:2021 (style 6J-M8C without shielding) [6] with the *periCORE Development Board*. The daughterboard is shown in Figure 25. The connector is intended for hybrid SPE cables with two wire pairs, where one pair is used for the power supply, and the second pair is a differential 100BASE-T1 pair (SPE). The face of the connector with the signal definitions is given in Figure 26.

Figure 25: M8H Female DaughterBoard

Contact	PMA signal	Wire colour
1	BI_DA+	Blue
2	BI_DA-	White
3	U	Red
4	GND	Black

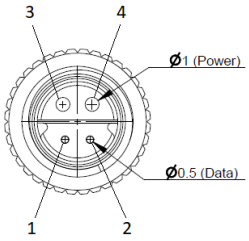


Figure 26: M8H Female connector drawing and pinout [16]

The pinout of the M8H Female Daughterboard (see Figure 27) is:

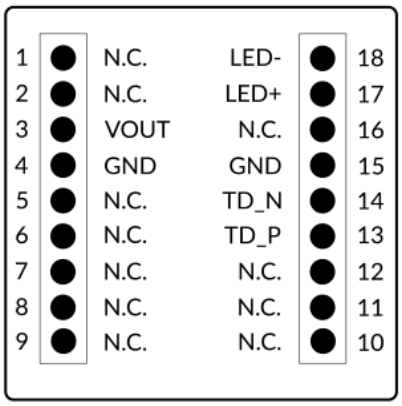
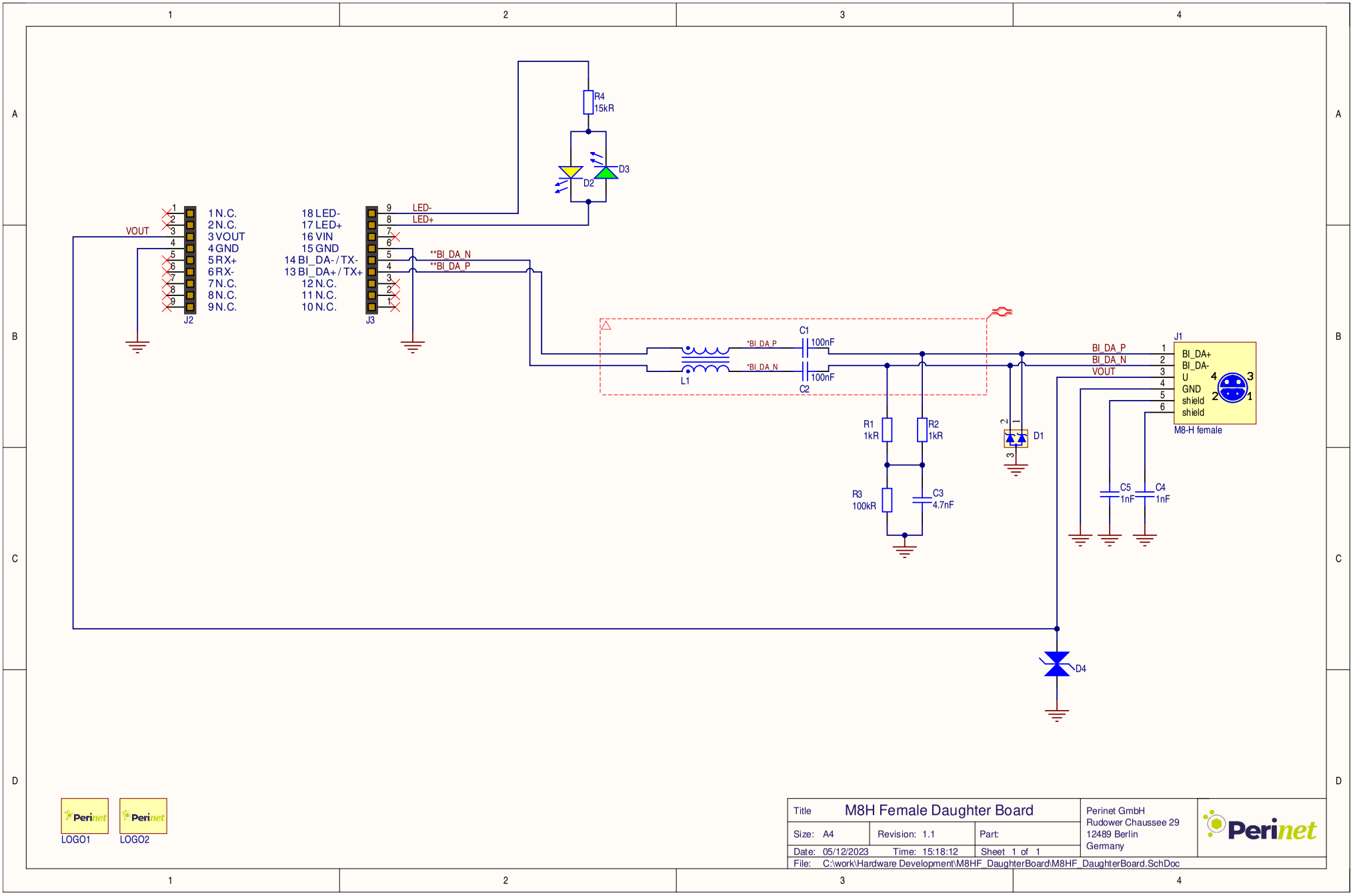


Figure 27: M8H Female DaughterBoard pinout

The schematic of the M8H Female DaughterBoard is given on the next page:

- Pin 3 of the M8H Female connector is connected with pin 3 of the daughterboard which means that the development board can supply power to devices connected with this daughterboard.
- There are two LEDs (D2 and D3) for showing various link status indicators. The indicators can be configured in software.
- M8H Female daughterboard is intended to be used on the 100BASE-T1 port which is configured as master.



LOGO1 LOGO2

Title	M8H Female Daughter Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part:	
Date: 05/12/2023	Time: 15:18:12	Sheet 1 of 1	
File: C:\work\Hardware Development\M8HF_DaughterBoard\M8HF_DaughterBoard.SchDoc			

B.3 T1 Industrial Jack DaughterBoard

The T1 Industrial Jack DaughterBoard allows using the *HARTING* T1 Industrial Jack connector which is defined in the IEC 63171-6:2020 [6] standard, under style 2J-L, with the *periCORE Development Board*. This type of connector is dedicated for SPE, and it is recommended for industrial applications. The daughterboard is shown in Figure 28.

Figure 28: T1 Industrial Jack DaughterBoard

The pinout of the T1 Industrial Jack DaughterBoard is shown in Figure 29.

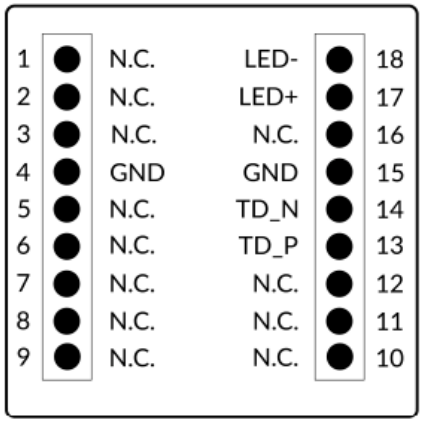
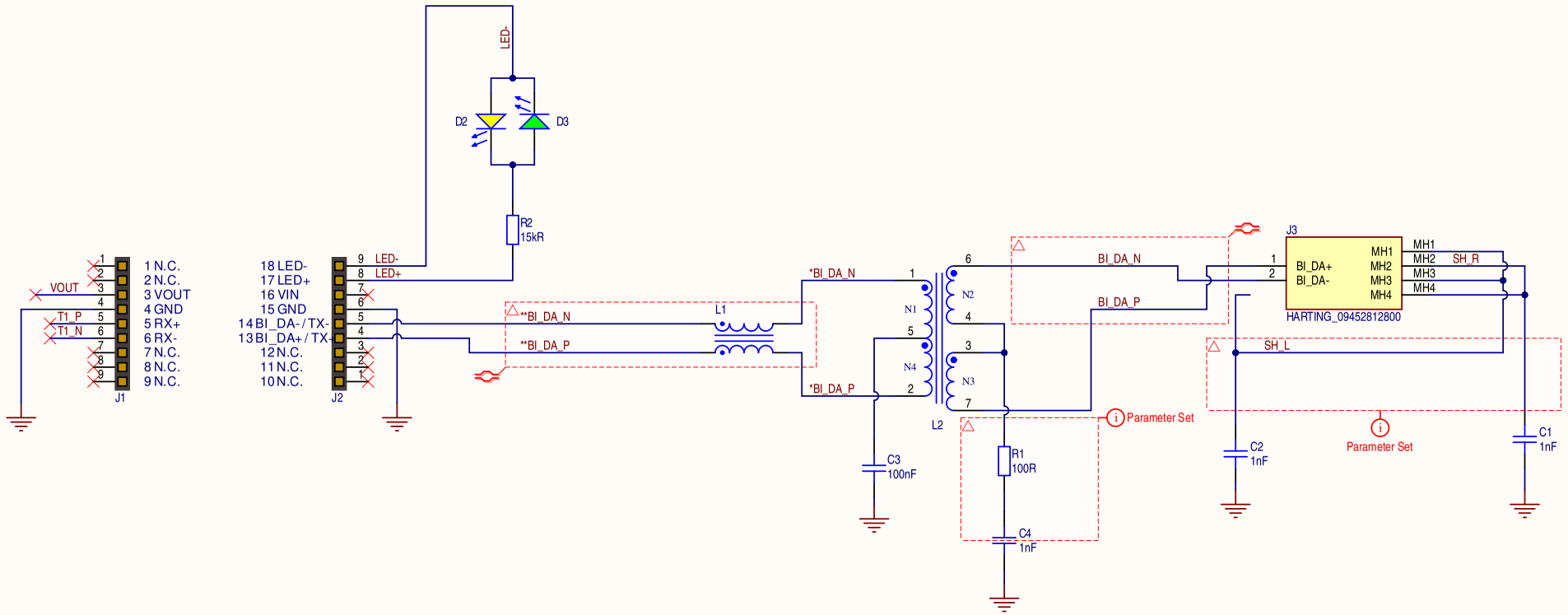


Figure 29: T1 Industrial Jack DaughterBoard pinout

The schematic of the T1 Industrial Jack DaughterBoard is shown on the next page:

- The design uses a transformer in order to provide the isolation of 1.5 kV between the development board (*periCORE*) and the signals at the connector.
- There are two LEDs (D2 and D3) for showing various link status indicators. The indicators can be configured in software.



Perinet LOGO1

Perinet LOGO2

Title	T1 Industrial Jack DaughterBoard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 05/12/2023	Time: 15:20:31	Sheet 1 of 1	
File: C:\work\Hardware Development\T1_Industrial_Jack_AH_DaughterBoard\T1_Industrial_Jack_AH_DaughterBoard.SchDoc			

B.4 T1H DaughterBoard

The T1H DaughterBoard is intended for the development of home automation applications, where cables with open ends are most common. This daughterboard has 4 spring-clamp terminal blocks produced by METZ CONNECT. Figure 30 shows the T1H DaughterBoard.

Figure 30: T1H DaughterBoard.

The colour of the clamps, as well as the names of the signals, comply with the SPE INDUSTRIAL PARTNER NETWORK application note [16]:

- GND (Black) - power supply negative terminal
- U (Red) - power supply positive terminal
- BI_DA- (White) - SPE differential pair negative signal
- BI_DA+ (Blue) - SPE differential pair positive signal

The pinout of the T1H DaughterBoard is shown in Figure 31.

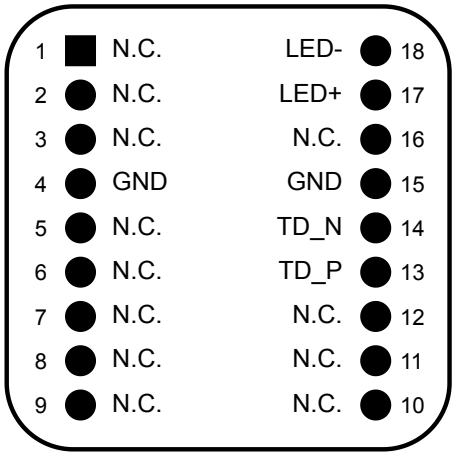
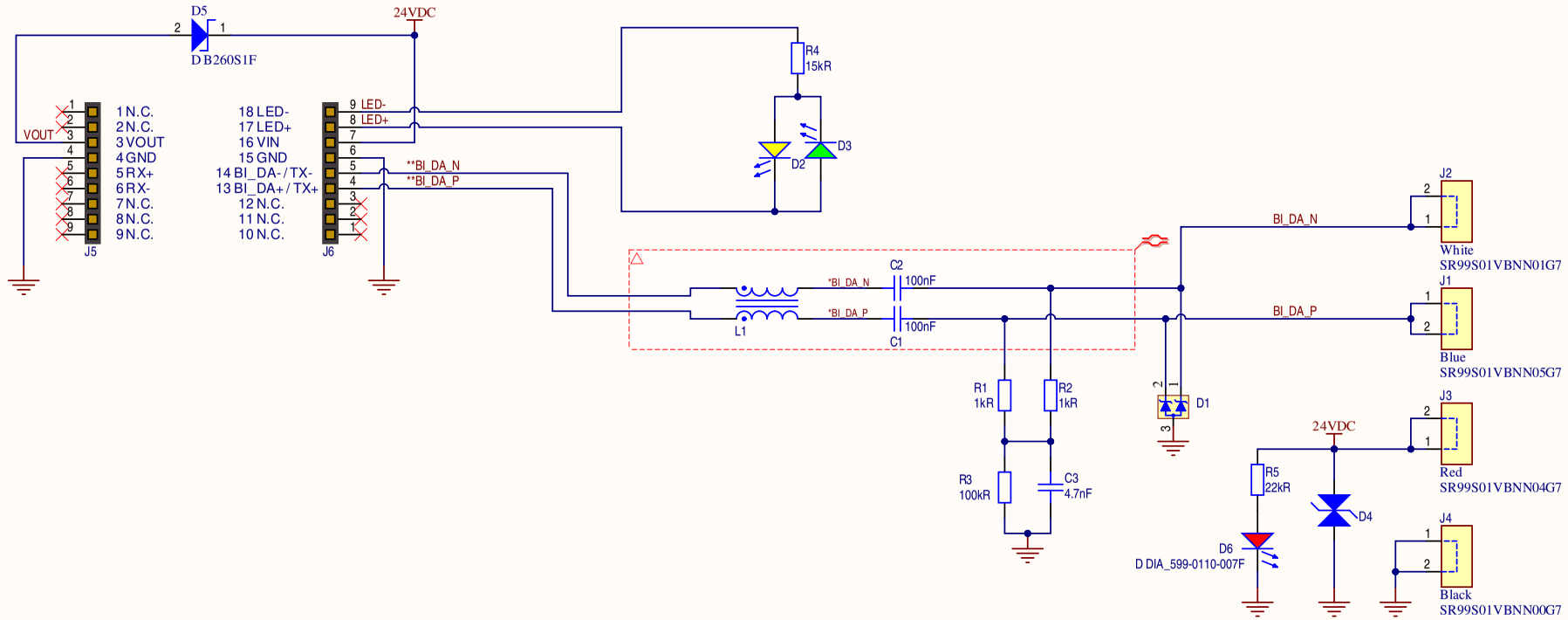


Figure 31: T1H DaughterBoard pinout

The schematic of the T1H DaughterBoard is shown on the next page:

- Depending on the wiring, this daughterboard can supply power to the development board, or source power to the next node on the network.
- There are two LEDs (D2 and D3) for showing various link status indicators. The indicators can be configured in software.
- LED D6 indicates the presence of the input power supply on the board



LOGO1 LOGO2

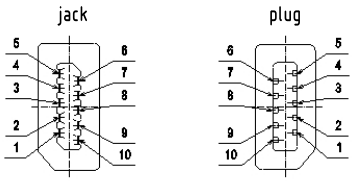
Title	T1H Daughter Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.0	Part: Connection	
Date: 05/12/2023	Time: 15:28:50	Sheet 1 of 1	
File: C:\work\Hardware Development\T1H_DaughterBoard\T1H_DaughterBoard.SchDoc			

B.5 ix DaughterBoard

The ix DaughterBoard is a network daughterboard which allows using the ix (A-coded) connector from *HARTING* with the *periCORE Development Board*. The ix connector is an industrial Ethernet connector defined in the IEC 61076-3-124 standard [7], and it supports 100BASE-TX. Figure 32 shows the ix DaughterBoard.

Figure 32: ix DaughterBoard

The drawing of the face of the connector as well as its pinout is shown in Figure 33.



Pin No. ix	10BASE-T 100BASE-TX	1/10GBASE-T	EIA/TIA 568A	EIA/TIA 568B	Industrial (PROFINET)
1	TX+	BI_DA+	white/green	white/orange	yellow
2	TX-	BI_DA-	green	orange	orange
3	N.C.	GND	-	-	-
4	N.C.	BI_DC+	blue	blue	-
5	N.C.	BI_DC-	white/blue	white/blue	-
6	RX+	BI_DB+	white/orange	white/green	white
7	RX-	BI_DB-	orange	green	blue
8	N.C.	GND	-	-	-
9	N.C.	BI_DD+	white/brown	white/brown	-
10	N.C.	BI_DD-	brown	brown	-

Figure 33: ix DaughterBoard pin assignment

The pinout of the daughterboard is shown in Figure 34

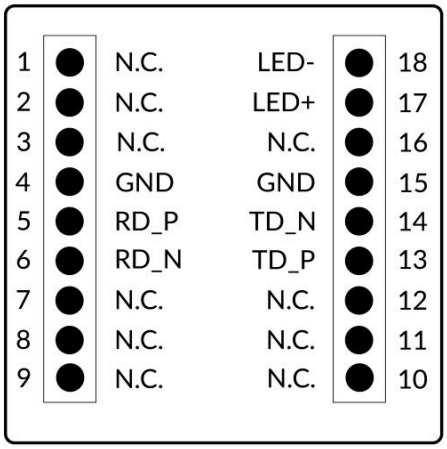
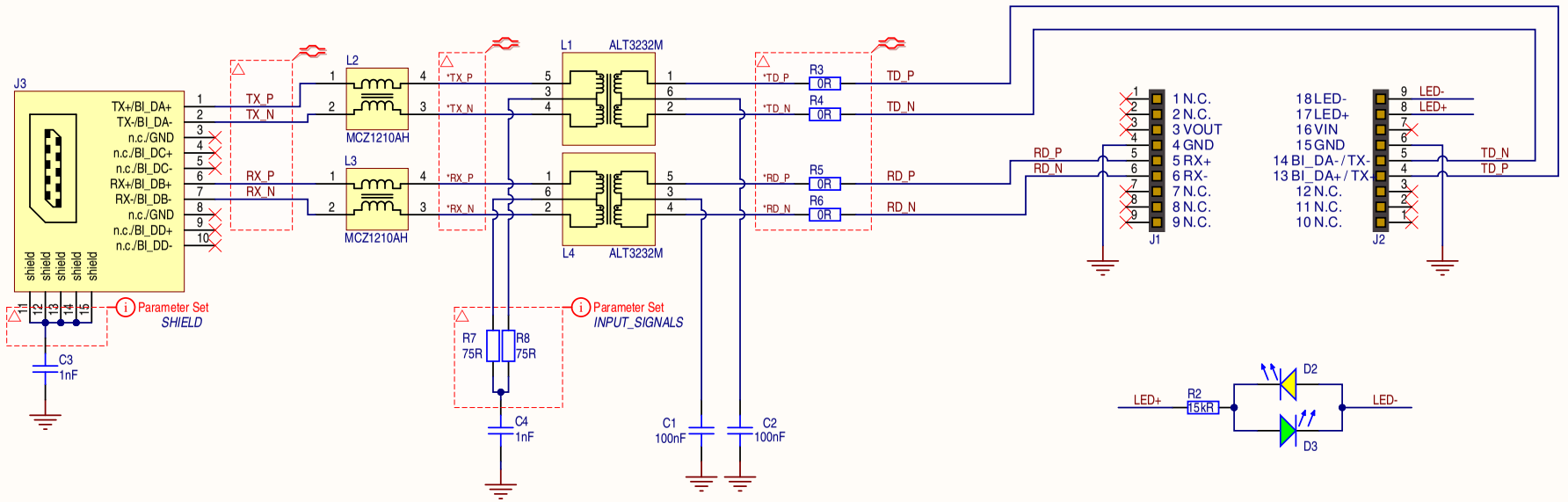


Figure 34: ix DaughterBoard pinout

The schematic of the ix DaughterBoard is shown on the next page:

- The board uses an A-coded ix connector.
- The ix DaughterBoard uses 100BASE-TX and is intended for `PORT4` of the development board.
- There are two LEDs (`D2` and `D3`) for showing various link status indicators. The indicators can be configured in software.



LOGO1

LOGO2

Title	ix DaughterBoard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 05/12/2023	Time: 15:53:25	Sheet 1 of 1	
File: C:\work\Hardware Development\iX_DaughterBOard\iX_DaughterBOard.SchDoc			

B.6 RJ45 DaughterBoard

The RJ45 DaughterBoard is a network daughterboard which allows using the RJ45 connector with the *periCORE Development Board*. This network daughterboard is for 100BASE-TX, therefore it should be used on the Port 4 on the *periCORE Development Board*. The daughterboard is shown in Figure 35.

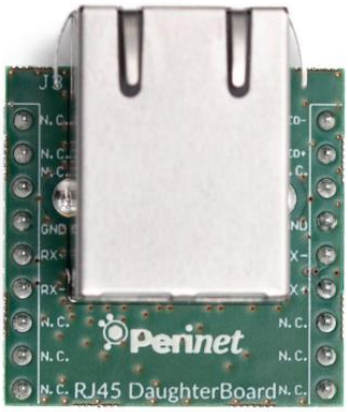


Figure 35: RJ45 DaughterBoard

The pinout of the RJ45 DaughterBoard is shown in Figure 36.

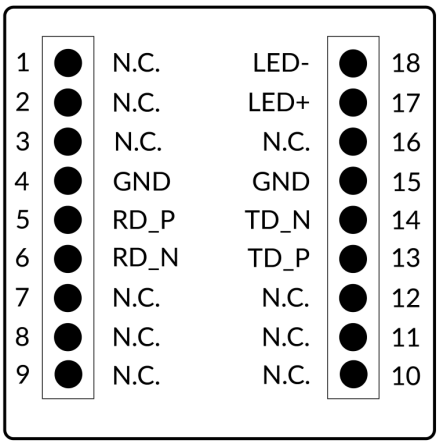
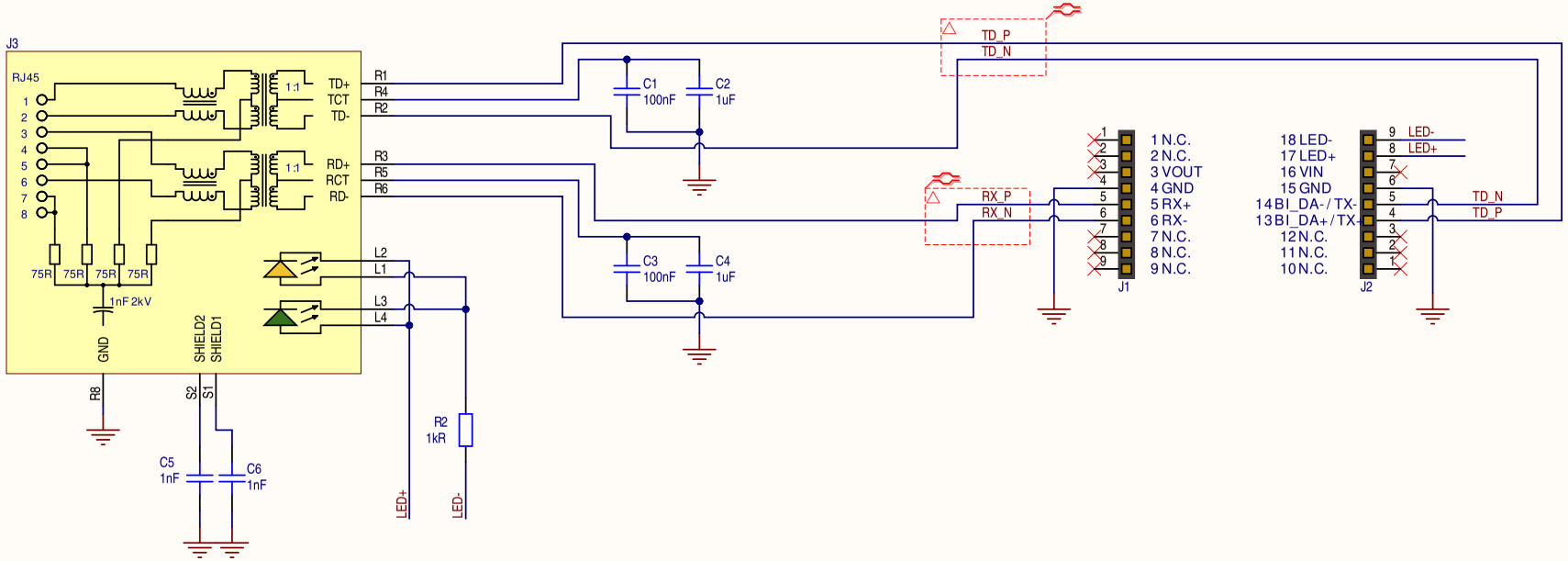


Figure 36: RJ45 DaughterBoard pinout

The schematic of the RJ45 DaughterBoard is shown on the next page.



LOGO1 LOGO2

Title	RJ45 Daughterboard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 05/12/2023	Time: 15:42:13	Sheet 1 of 1	
File: C:\work\Hardware Development\RJ45_DaughterBoard\RJ45_DaughterBoard.SchDoc			

C Sensor/Actuator daughterboards

C.1 0-10V DaughterBoard

The 0-10V DaughterBoard (Figure 37) allows connecting sensors with 0-10V analog output signal to the *periCORE* development board. The board is based on the ADS1113 ADC from TI. The communication interface between the *periCORE* module and the ADC is I^2C.

Figure 37: 0-10V DaughterBoard

The daughterboard has a 4-pin male terminal block from HARTING (har-flexicon 14120414002000). The daughterboard comes with the female counterpart, har-flexicon 14312301201, which can be used for connecting the sensor cable. The names of the signals are printed on the daughterboard's silkscreen. The following signals are used:

- *GND* - ground connection
- *IN* - input voltage (0-10V)
- *TCH* — open-drain output which can be used to control the calibration (teach) process of some sensors. It is driven by GPIO1 of the *periCORE* module
- *24V* — power supply positive terminal, for supplying power to the sensor

A typical wiring example is shown in Figure 38.

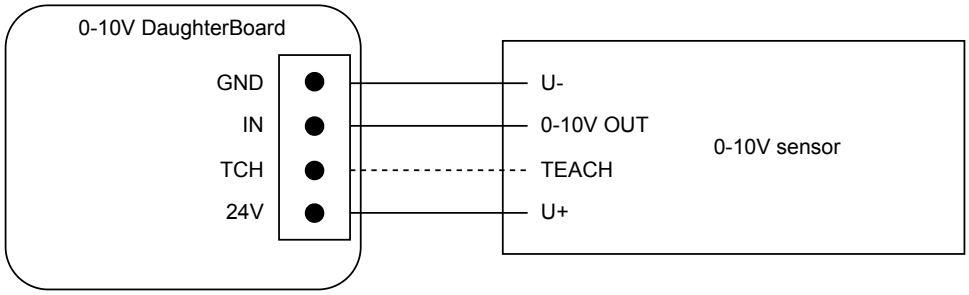
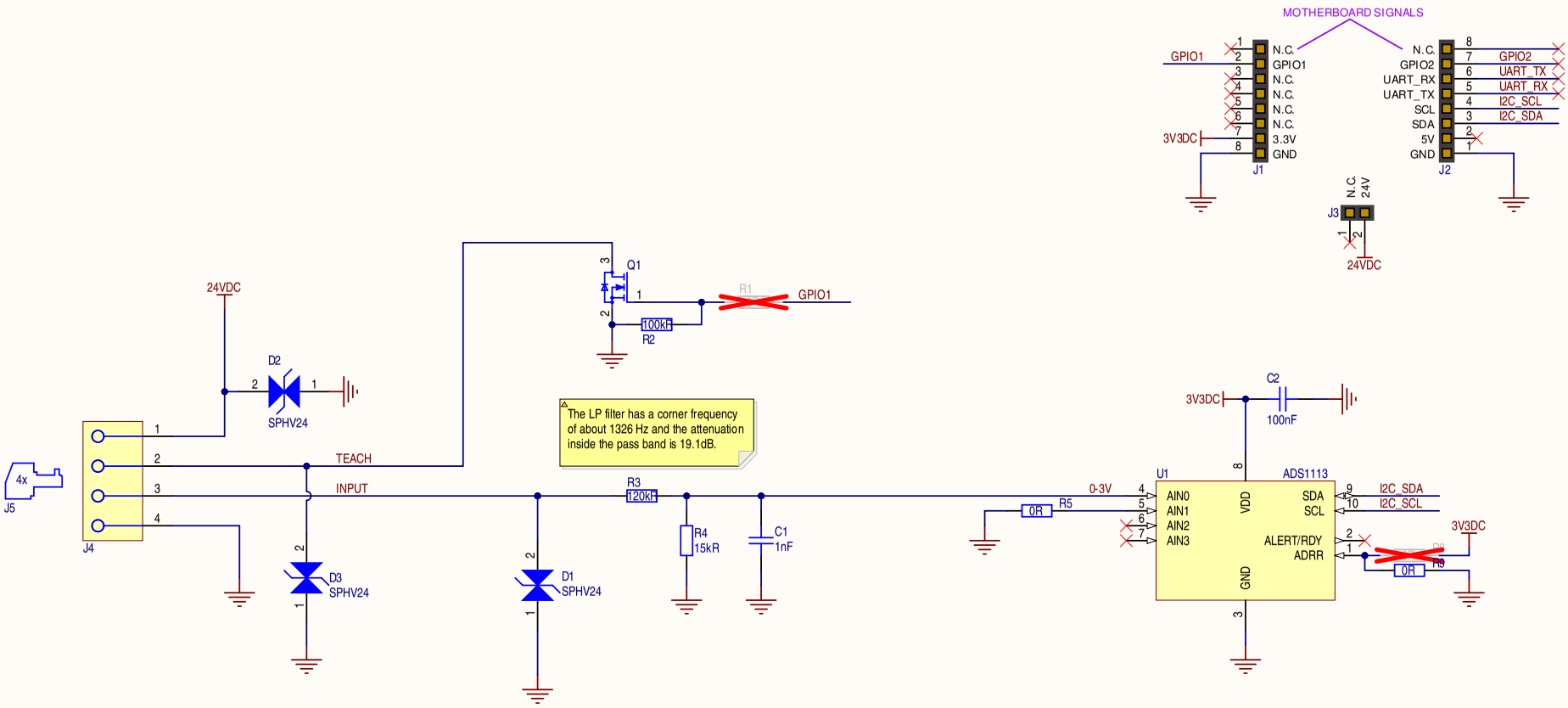


Figure 38: 0-10V DaughterBoard

The schematic of the 0-10V DaughterBoard is given on the next page. The main features of the circuit are:

- The input voltage is attenuated through the voltage divider R3 - R4, C1 by the factor of 9, with -3dB point at about 1326 Hz.
- The address of the ADC on the I^2C bus is 0x48.
- For controlling transistor Q1, which can be used for the calibration (teach) of some sensors, resistor R1 should be populated with the value of about 1kΩ.



Title	0-10V DaughterBoard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 05/12/2023	Time: 15:55:11	Sheet 2 of 2	
File: C:\work\Hardware Development\010V_DaughterBoard\010V_DaughterBoard.SchDoc			

C.2 PT100 DaughterBoard

The PT100 DaughterBoard (Figure 39) allows the development of smart PT100 sensors with the *periCORE* module. The daughterboard uses ADS112C04, a 16-bit ADC from Texas Instruments, as an analog frontend and interface between the *periCORE* and the PT100. The controlling and reading of the digital samples from the ADC is over the I^2C bus. The PT100 DaughterBoard is designed to be used with 3-wire PT100 sensors.

Figure 39: PT100 DaughterBoard

The wiring diagram is shown in Figure 40.

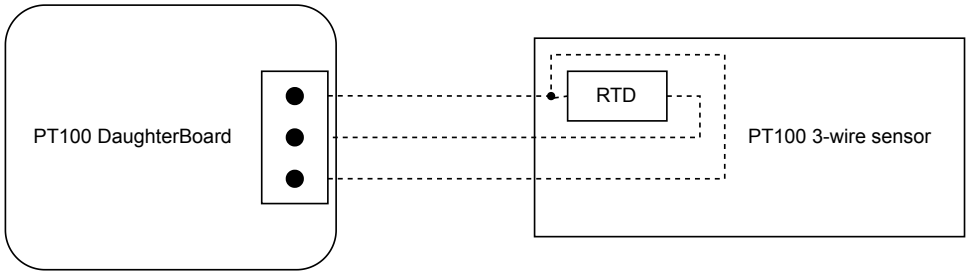
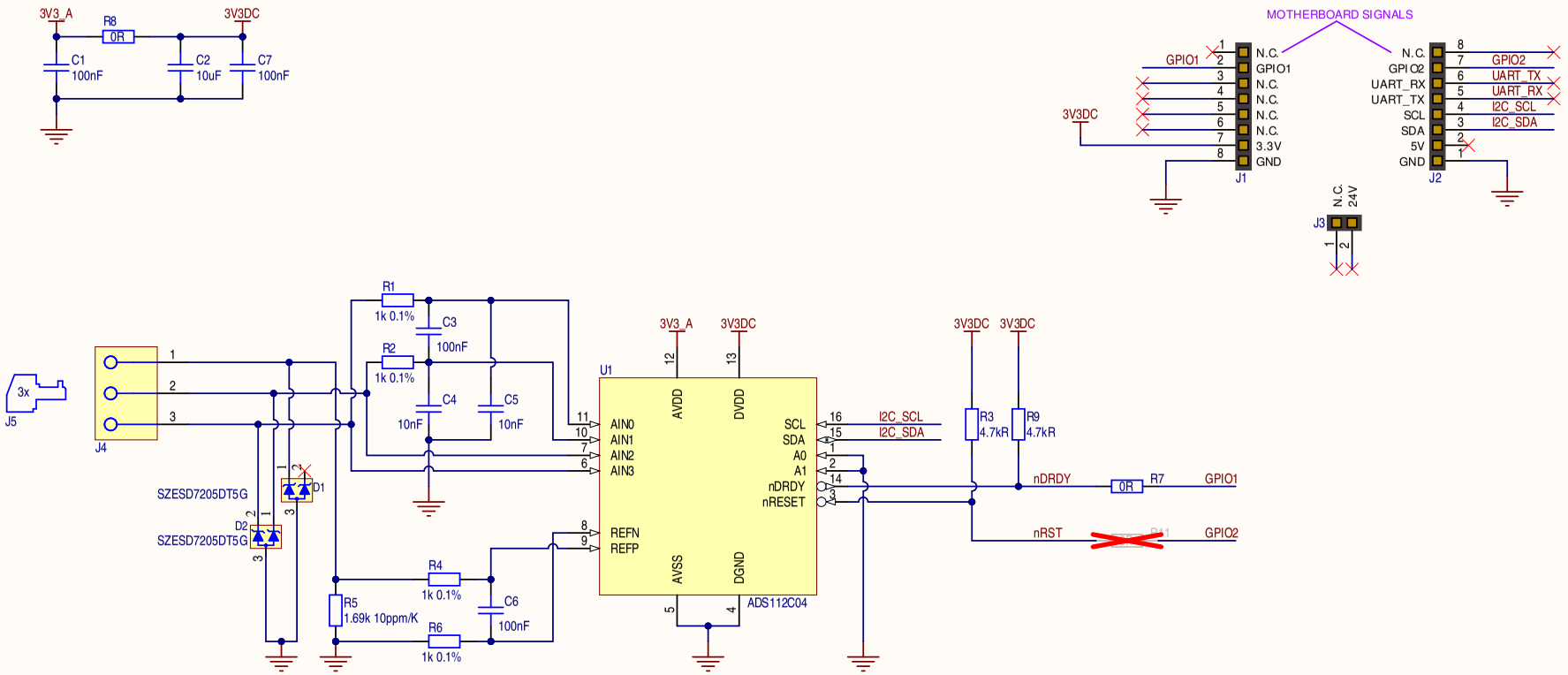


Figure 40: PT100 Wiring Diagram

The schematic of the PT100 DaughterBoard is shown on the next page. The main features of the circuit are:

- The excitation current of the PT100 element is 500μA
- Maximum wire resistance is 15Ω
- Temperature measurement range: -200°C to +850°C



Title	PT100 Daughter Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 05/12/2023	Time: 15:57:35	Sheet 2 of 2	
File: C:\work\Hardware Development\PT100_DaughterBoard\PT100_DaughterBoard.SchDoc			

LOGO1 LOGO2

C.3 GPIO DaughterBoard

The GPIO DaughterBoard (Figure 41) allows development of sensor/actuator applications that require 24V digital I/O and is functionally identical to the *periNODE-GPIO*. It has two I/O channels and is based on the MAX22515 IC from Analog Devices, controlled by the *periCORE* module via the I^2C bus.

Figure 41: GPIO DaughterBoard

The schematic of the GPIO DaughterBoard is shown on the next page. The board has a 4-pin male terminal block (har-flexicon 14120314002000), *J1*, which is used to connect the IO signals to the daughterboard. Its female counterpart har-flexicon 14310314402000 is included in the package. The names of the signals are printed on the board overlay:

- 24V — power supply positive terminal, can be used to supply an external circuit
- IO1 — IO channel 1; can be configured in firmware for input or output function
- IO2 — IO channel 2; can be configured in firmware for input or output function
- GND — electrical ground

An example wiring is shown in Figure 42. The motor is supplied with power from the GPIO DaughterBoard. The up/down movement of the motor is controlled with the I/O channels IO1 and IO2 which are configured for output operation.

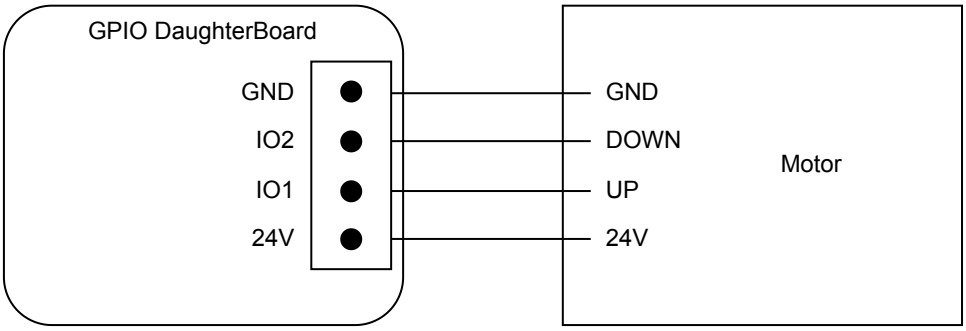
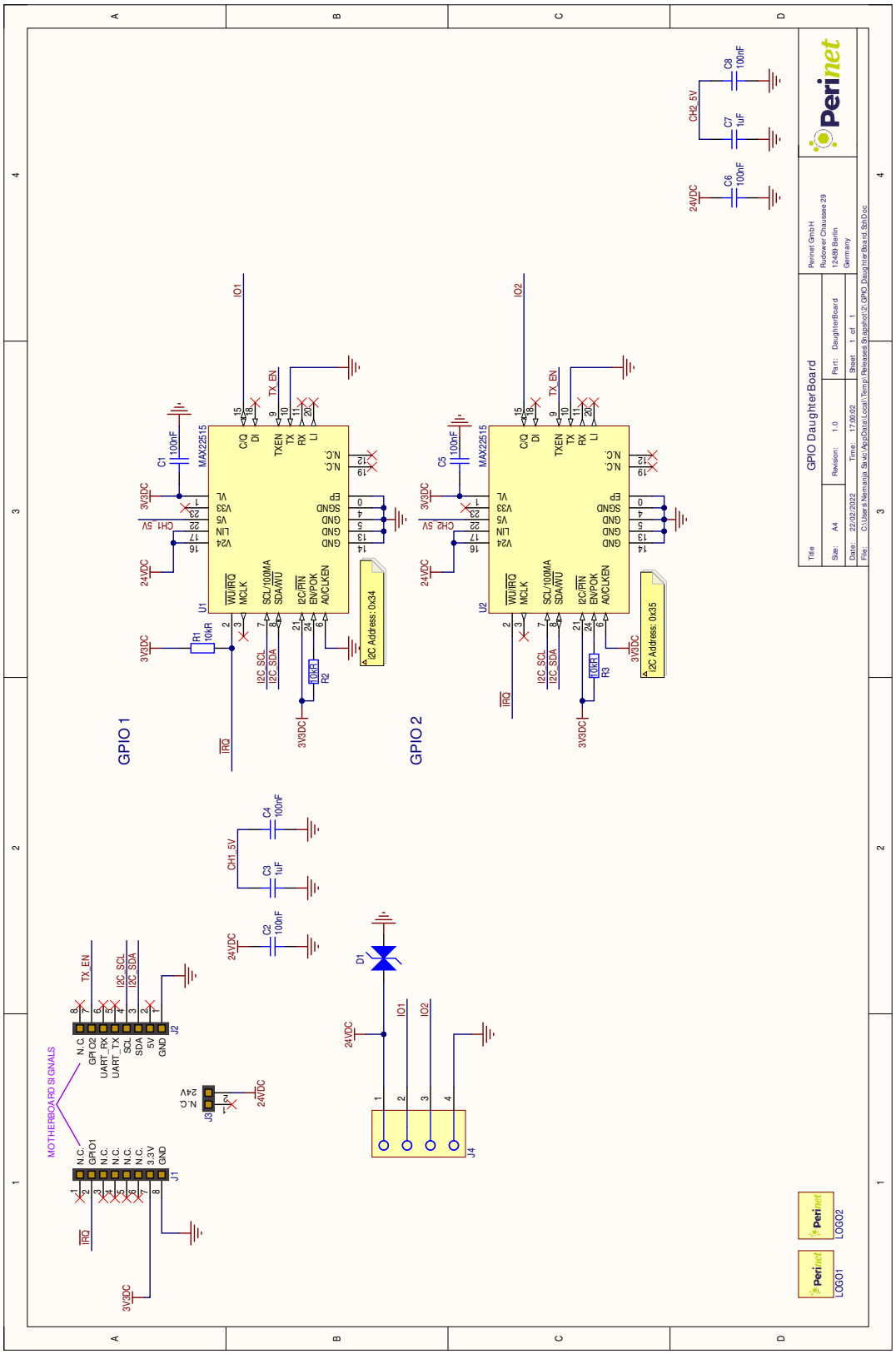


Figure 42: GPIO DaughterBoard wiring example


GPIO DaughterBoard
GPIO 1
GPIO 2
MOTHERBOARD SIGNALS
MAX22515
I2C Address: 0x34
I2C Address: 0x35
3V3DC
24VDC
5V
IRQ
I2C_SCL
I2C_SDA
TX_EN
IO1
IO2
Perinet GmbH
Rudower Chaussee 29
12489 Berlin
Germany

C.4 SPI DaughterBoard

The SPI Daughterboard (Figure 43) supports development with the SPI interface of the peri-CORE. The SPI feature of the periCORE uses the I^2C bus and both GPIO pins to provide a software SPI implementation. The SPI Daughterboard simply reroutes the pins for use with click boards; the pinout is based on the mikroBUS pin description for click boards from MikroE (see the pin-description section).

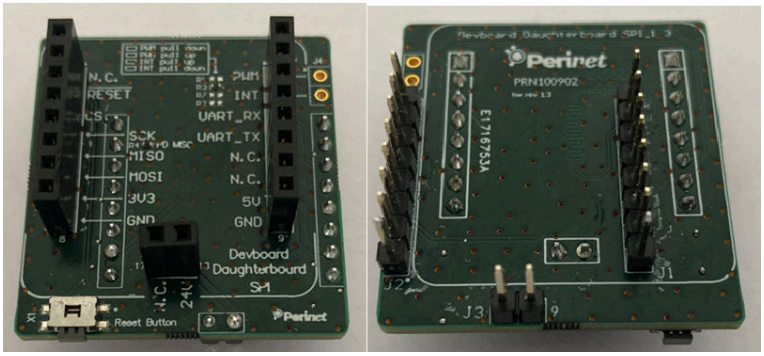


Figure 43: SPI DaughterBoard

The typical wiring is shown in Figure 44. The SPI Daughterboard supports SPI and UART, which are routed directly from the periCORE. The 3.3V, 5V and 24V supplies are also routed from the development board to the SPI Daughterboard. Only the “PWM” and “INT” pins are not routed to the development board; they can be accessed by adding an additional pin header on J4. The reset pin is directly connected to the mechanical switch (if available) on the SPI Daughterboard and requires a pull-up on the click-board side.

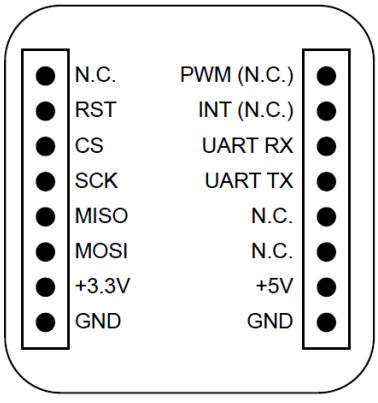
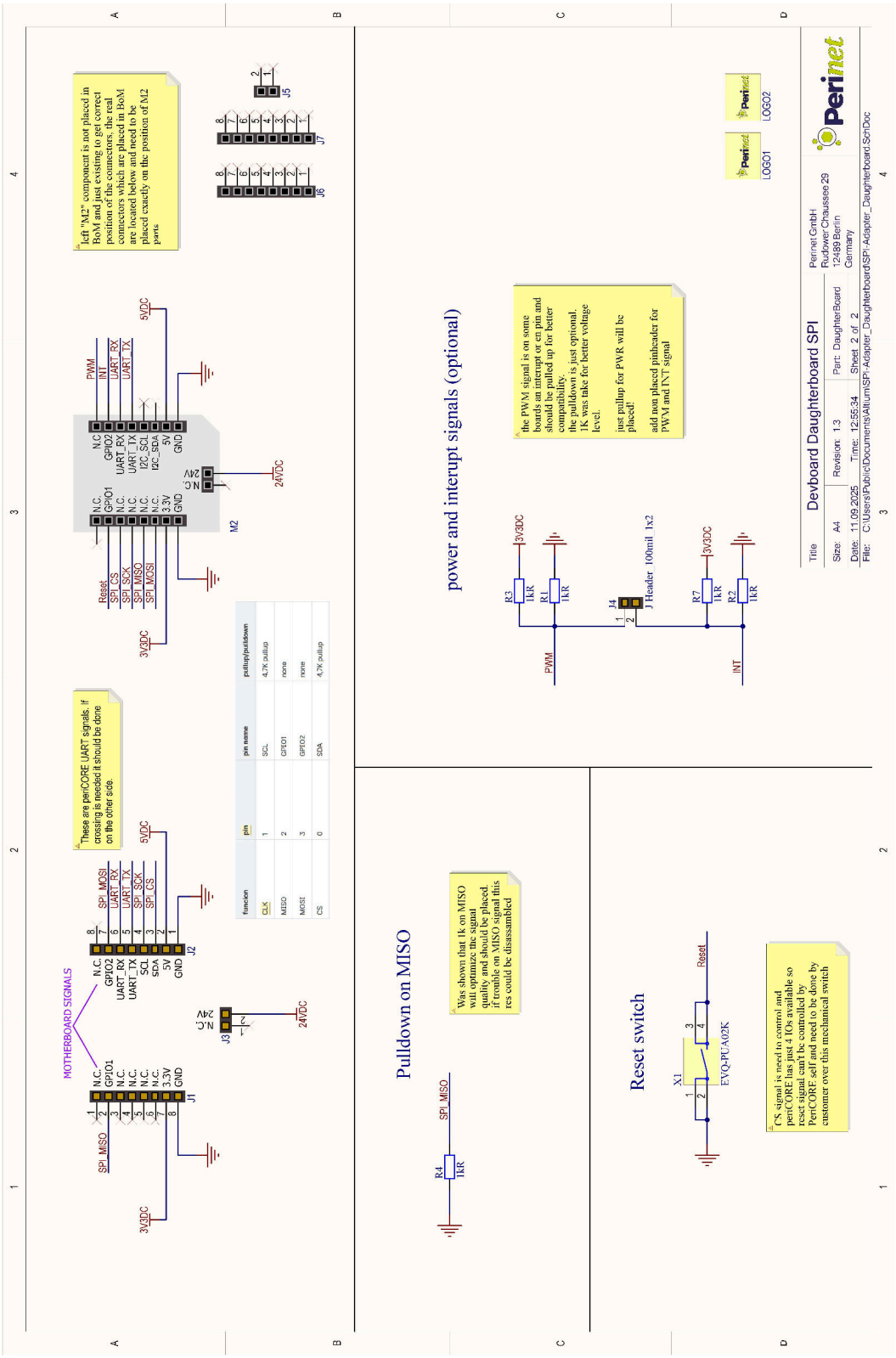


Figure 44: SPI DaughterBoard wiring example


MOTHERBOARD SIGNALS
These are periCORE UART signals. If crossing is needed it should be done on the other side.
left "M2" component is not placed in BoM and just existing to get correct position of the connectors. the real connectors which are placed in BoM are located below and need to be placed exactly on the position of M2 parts
power and interupt signals (optional)
the PWM signal is on some boards an interupt or an pin and should be pulled up for better compatibility.
the pulldown is just optional. 1K was take for better voltage level.
just pullup for PWR will be placed!
add non placed pinheader for PWM and INT signal
Pulldown on MISO
Was shown that 1k on MISO will optimize the signal quality and should be placed. if trouble on MISO signal this res could be disassambled
Reset switch
CS signal is need in control and periCORE has just 4 IOs available so reset signal can't be controlled by PeriCORE self and need to be done by customer over this mechanical switch
Devboard Daughterboard SPI
Perinet GmbH
Rudower Chaussee 29
12489 Berlin
Germany
Size: A4
Revision: 1.3
Part: DaughterBoard
Date: 11.09.2025
Time: 12:55:34
Sheet 2 of 2
File: C:\Users\Public\Documents\Altium\SPI-Adapter_Daughterboard\SPI-Adapter_Daughterboard.SchDoc

D List of Figures

E List of Listings

F List of Tables

G Glossary

100BASE-T1 A Ethernet Standard where two endpoints are connected by a single twisted pair cable. It is one of the so-called Single Pair Ethernet (SPE) standards. It operates in full-duplex with a data rate of 100 MBit per second. Furthermore, it uses PAM-3 modulation with a voltage level from -1 to +1V, differentially on the two wires. 11, 46, 47, 49, 52

100BASE-TX A Ethernet Standard where two twisted pairs with differential signals are used, one for each direction. The data rate is 100 MBit per second. It is also called ”Fast Ethernet”. 11, 46, 47, 60

ADC Analog Digital Converter. 67

API Application Programming Interface. 3, 17, 21–23, 28, 33

Base64 Base64 allows representation of binary data in text format. 28, 33, 76

css Cascading Style Sheets. 28, 32, 33, 76

DC Direct current. 10

DNS-SD DNS Service Discovery [4] is a way of using standard DNS programming interfaces, servers and packet formats to browse the network for services. 8, 23

DOM Document Object Model. 29–31, 76

GND ground, zero potential mass. 9

html HyperText Markup Language. 28–30, 33, 76

HTTP Hypertext Transfer Protocol is an application-layer protocol for transmitting hypermedia documents, such as HTML. 17, 21–28, 33, 34, 74, 76

I2C Inter-Integrated Circuit is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial bus. 10, 14

IIoT Industrial Internet of Things. 5, 6

IPv4 Internet Protocol Version 4, a communication protocol. 34

IPv6 Internet Protocol Version 6 [8], a communication protocol. 22, 23, 34

jpeg An image file format. 33

js JavaScript. 28, 31–33

JSON JavaScript Object Notation is standard text-based format for representing structured data based on JavaScript object syntax [3]. 18, 24, 25, 27

JTAG Joint Test Action Group is a standard that defines tools to debug embedded systems. 9, 13

LED Light-Emitting Diode is a semiconductor that emits light when current flows through it. 10, 46, 52, 54, 57, 60

LLMNR The Link-Local Multicast Name Resolution is a protocol based on the Domain Name System packet format that allows IPv6 hosts to perform name resolution for hosts on the same local link. 23

Makefile Definition of set of tasks to be executed by GNU make tool. 28, 33

MDI Media Dependent Interface, a Fast-Ethernet chipset component. 11

mDNS multicast Domain Name Service [5], a protocol that implements a local distributed name resolving mechanism. 8, 10, 18, 19, 23, 34, 74, 76

MQTT Message Queuing Telemetry Transport is a lightweight, publish-subscribe based network protocol that transports messages between devices. 17, 21–23, 26, 27, 74

mTLS Mutual TLS extends the TLS protocol by requiring clients to pass certificates, allowing to provide authorization mechanisms of Application services. 23, 25, 28

OS Operating System, a system software that manages computer hardware and software resources. 20, 34

PC Personal Computer. 16

PCB Printed Circuit Board. 10

PHY Physical layer. Lowest layer of the OSI model. 46

png Portable Network Graphics - an image file format. 28, 33

RD Receive Data port. 46, 47

REST REpresentational State Transfer, a web API style. 17, 22, 23, 28, 33

RMII Reduced Media-Independent Interface, an interface defined as a standard interface to connect a media access control (MAC) block to an Ethernet physical transceiver chip. 9, 11

SDK Software Development Kit is a colletion of development tools. 6

SPE Single Pair Ethernet. 5, 10, 11, 17, 47, 51, 54

SPI The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for inter IC communication. 17

SSH Secure Shell Protocol. 19, 74

TCP Transmission Control Protocol. 17–20, 34

TD Transmit Data port. 46, 47

TLS Transport Layer Security Protocol. Used by Application Protocols like MQTT or HTTP to allow secure data transfer. 23, 28

UART A Universal Asynchronous Receiver-Transmitter is a computer hardware device for asynchronous serial communication. 3, 10, 14, 17, 19, 20

UI User Interface. 6, 19, 21, 22, 28, 29, 31, 33, 35, 74

URI Uniform Resource Identifier. 23–25, 32

USB Universal Serial Bus is a standard of the connection of peripherals to personal computers. 17, 18

H References

[1] URL: https://www.mikroe.com/sht-click.

[2] ARM. *ARM Standard JTAG*. URL: https://developer.arm.com/documentation/101416/0100/Hardware-Description/Target-Interfaces/ARM-Standard-JTAG.

[3] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259.

[4] Stuart Cheshire and Marc Krochmal. *DNS-Based Service Discovery*. RFC 6763. Feb. 2013. DOI: 10.17487/RFC6763. URL: https://www.rfc-editor.org/info/rfc6763.

[5] Stuart Cheshire and Marc Krochmal. *Multicast DNS*. RFC 6762. Feb. 2013. DOI: 10.17487/RFC6762. URL: https://www.rfc-editor.org/info/rfc6762.

[6] "Connectors for electrical and electronic components - Product requirements - Part 6: Connectors - Detail specification for 2-way and 4-way (data/power), shielded, free and fixed connectors for transmission capability and power supply capability with frequencies up to 600 MHz". In: *IEC 63171-6* (2021). URL: https://webstore.iec.ch/publication/68051.

[7] "Connectors for electrical and electronic equipment - Product requirements - Part 3-124: Rectangular connectors - Detail specification for 10-way, shielded, free and fixed connectors for I/O and data transmission with frequencies up to 500 MHz". In: *IEC 61076-3-124:2019* (2019). URL: https://www.vde-verlag.de/iec-normen/247243/iec-61076-3-124-2019.html.

[8] Dr. Steve E. Deering and Bob Hinden. *IP Version 6 Addressing Architecture*. RFC 4291. Feb. 2006. DOI: 10.17487/RFC4291. URL: https://www.rfc-editor.org/info/rfc4291.

[9] Perinet GmbH. periCORE based firmware example for the periNODE-0-10V. URL: https://gitlab.com/perinet/periNODE-0-10V/periNODE-distance-firmware.git.

[10] Perinet GmbH. periCORE Datasheet. PRN.100.375. https://docs.perinet.io/PRN100375-periCOREDatasheet.pdf.

[11] Perinet GmbH. periCORE Development Kit Setup Application Note. PRN.100.376. https://docs.perinet.io/PRN100376-periCOREDevelopmentKitSetupApplicationNote.pdf.

[12] Perinet GmbH. periCORE Firmware Development Application Note. PRN.100.379. https://docs.perinet.io/.

[13] Perinet GmbH. periMICA User Guide. PRN.100.392. https://docs.perinet.io/PRN100392-periMICAUserGuide.pdf.

[14] Perinet GmbH. sève Operating System Datasheet. PRN.100.377. https://docs.perinet.io/.

[15] MikroElektronika. *mikroBUS™ standard specification*. URL: https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf.

[16] SPE INDUSTRIAL PARTNER NETWORK. *SINGLE PAIR ETHERNET – M8 HYBRID SYSTEM*. URL: https://www.single-pair-ethernet.com/sites/default/files/2020-09/2020-09_spe-applikationnote_spe-hybrid_v10en.pdf.

[17] Karsten Walther. "INTERNET-OF-THINGS MODULE". European pat. 4038350. Perinet GmbH. Sept. 30, 2019. URL: http://v3.espacenet.com/textdoc?IDX=EP4038350.

Revision History

Revision	Date	Author(s)	Description
1	2022-01-09	ckoe	Initial release: software development
2	2022-01-24	dper, ckoe, asch	Added hardware architecture. Updated software development section
3	2022-03-02	dshe	Add Overview section
4	2022-05-03	ckoe	Add Troubleshooting and System Architecture sections. Update all other sections with latest content.
5	2024-02-08	dber,nsav	Update URI's. Update Hardware architecture.
6	2026-05-04	tnau,shoe	add SPI daughterboard with all information; add Boot-Config Pins Section 2.5;