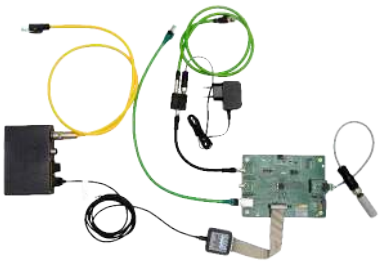
periCORE Development Kit

Full Featured Firmware Development Setup



User Guide

Perinet
Seamless IoT Connectivity

Doc-No.: PRN.100.378 rev: 4

May 3, 2022

Document Information

Title	periCORE Development Kit
Subtitle	Full Featured Firmware Development Setup
Type	User Guide
Status	Release
Version	4
Date	May 3, 2022
Disclosure Restriction	

Contents

1 Overview

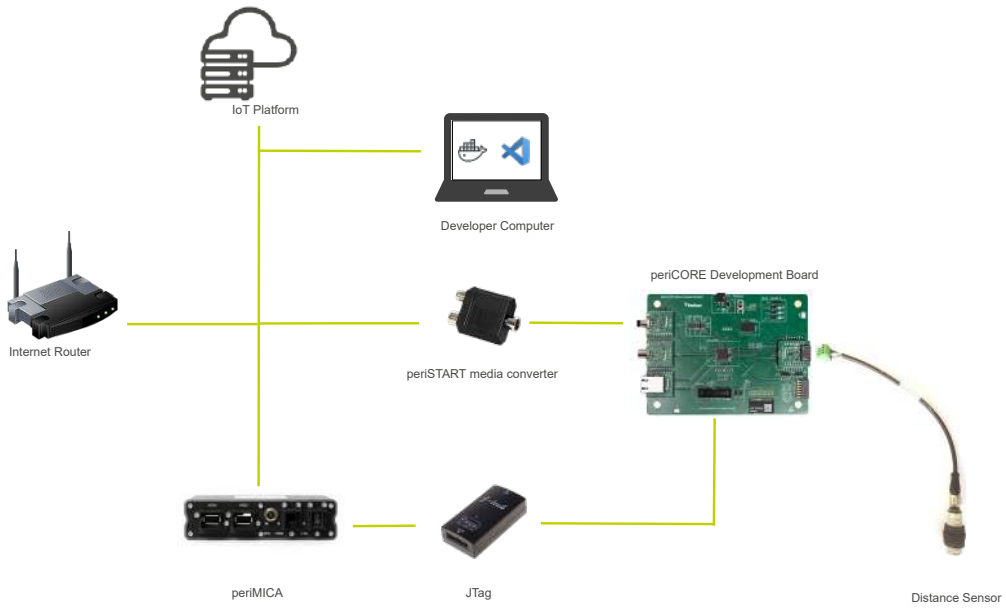


Figure 1: Development Kit Overview

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*. The purpose is to have an IIoT environment that allows to implement new applications, efficiently.

The first section (2) presents the *periCORE development board* hardware architecture. All the interfaces available in the module can be easily accessed using the development board peripherals, which is shown in here.

In the following section (3) the user will be guided to configure and use the software tools provided. It includes the *periCORE SDK* and the *pericoredbg Container* used to build and debug customized *periCORE* based applications without requiring to install any additional software which is a great benefit for the developers.

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

Besides that a powerful tool indirectly presented in the kit is the *periMICA* that can run typical field IIoT functions through its lightweight containers e.g. *MQTT Broker Container* (available for download in `https://perinet.io/downloads`).

Targeted Applications

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

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

Interfaces

- 2 x 100BASE-T1 Phy (IEEE 802.3bw)
- 1 x Combined 100BASE-T1/TX Phy
- 1 x MAC to arm processor core (Figure 2)
- 1 x UART
- 1 x I2C
- 2 x GPIO

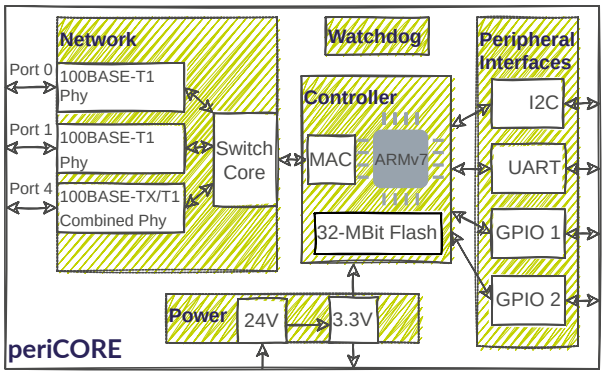


Figure 2: periCOREs hardware blocks.

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 3.8 mm (Figure 3)

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

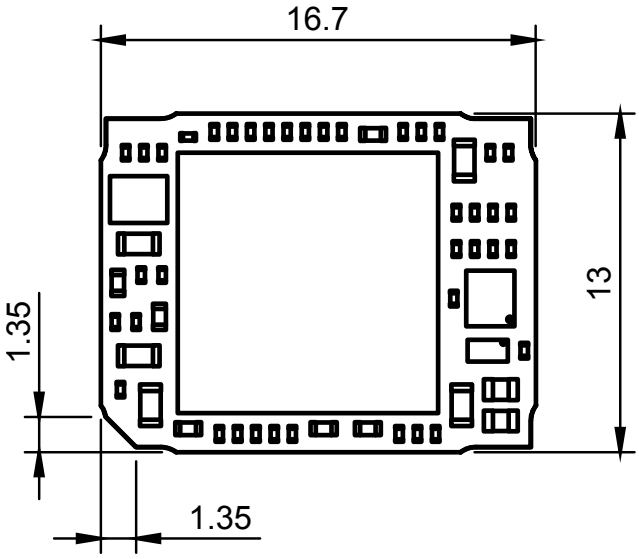


Figure 3: periCOREs dimensions in mm.

Compliance

- RoHS
- WEEE

Security

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

Software Library *libperiCORE*

- Rapid firmware development with *periCORE Development Kit* (see Figure 4)
- 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

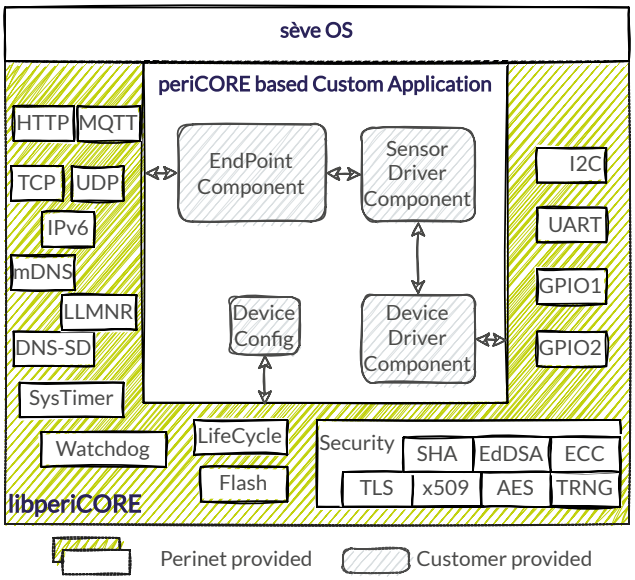


Figure 4: The software architecture with Custom Application template, provided by Perinet.

2 Hardware Architecture

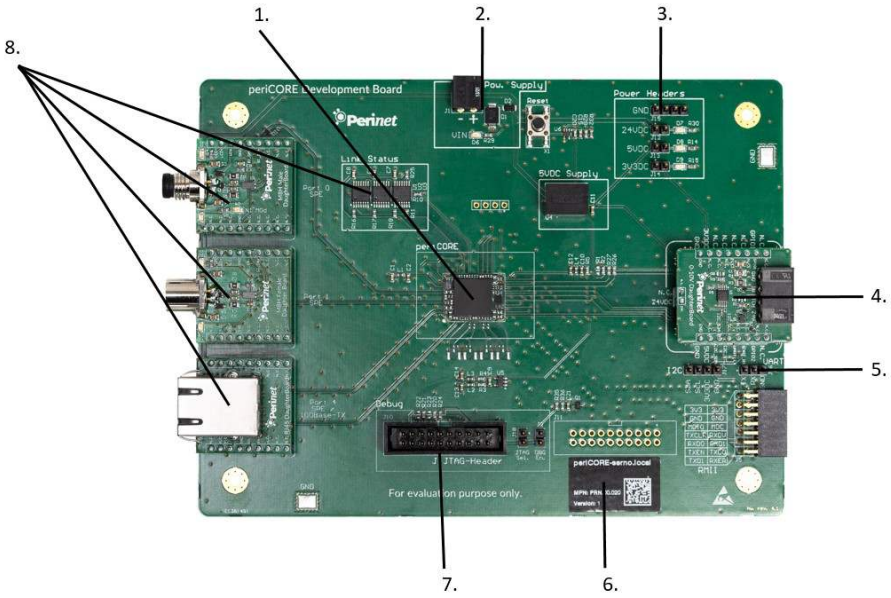


Figure 5: Top view on *periCORE development board* with all jumpers in default position, including all daughterboards.

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 board is kept modular from both network communication and sensor side in order to support a wide range of applications.

In this case, modularity means that the connectors for either network communication side and sensor side are provided by a separate daughterboard. This allows the user to evaluate multiple different network and sensor interfaces on the *periCORE development board*.

The relevant parts of the board are (5):

1. *periCORE* module
2. Input power connector (`J1`) and on-off button (`X1`)
3. Header connectors (`J12`, `J13`, `J14` and `J15`) with different voltage levels (24V, 5V, 3.3V and GND)
4. Sensor/actuator daughterboard (designated with `M2`)
5. Header connector (`J16`) for accessing UART signals
6. *periCORE* device label information with mDNS name.
7. JTAG connector (`J10`) for debug purposes

8. Network communication daughterboards (PORT0, PORT1, and PORT4) with network link status LED

Also see appendix A for all hardware schematics of the development board.

2.1 Power Supply

The nominal supply voltage for the development board is 24V DC which is internally converted to 3.3V. 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 different ways:

- via a PCB header connector (designated as J1 on the board)
- over the network cable where the power to the board comes via network communication daughterboards

2.1.1 Power over J1 connector

J1 is a male PCB header connector with a 3.5mm pitch. 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 network communication daughter board

One of the main features of the *periCORE* is the ability to daisy chain the sensor nodes from *Perinet Smart Components*, hence other *periCORE* based devices. This means that power and communication lines are coming to one node and going from that node to the next node, and so on. Development and prototyping of such applications is possible with this development board since the pins for incoming and outcoming power to and from the board are allocated in the footprint of the network communication daughterboard (see Figure 7).

If the power for the development board is provided by the network communication daughterboard the LED for indicating the presence of the input voltage should be implemented on the daughterboard itself. More information about the network communication daughterboards can be found in section 2.2.1

2.1.3 On-Off Button

Pressing the On-Off button (X1) switches the 24V_core voltage domain off (which subsequently switches 5V and 3.3V off). This button can be used for performing a power cycle of the periCORE and its peripherals.

2.2 Network Interfaces

periCORE supports the following Ethernet MDI:

- 100BASE-T1 – Ethernet over a single twisted pair
- 100BASE-TX – Ethernet over two twisted pairs

There are 3 ports for 100BASE-T1 and 100BASE-TX – `PORT0`, `PORT1`, and `PORT4` as can be seen under number 8 in Figure 5. `PORT0` and `PORT1` support only 100BASE-T1, while `PORT4` supports (only one at a time) both 100BASE-T1 and 100BASE-TX. The aforementioned ports are available on the development board in the form of slots where the user can place a daughterboard with an appropriate connector.

2.2.1 Network communication daughterboard

The network communication daughterboards are carrier boards for different connectors. Figure 6 shows the drawing of a daughterboard with M8 Hybrid Male connector. The daughterboard has two rows of male header pins which are fitted in the counterpart female headers on the development board.

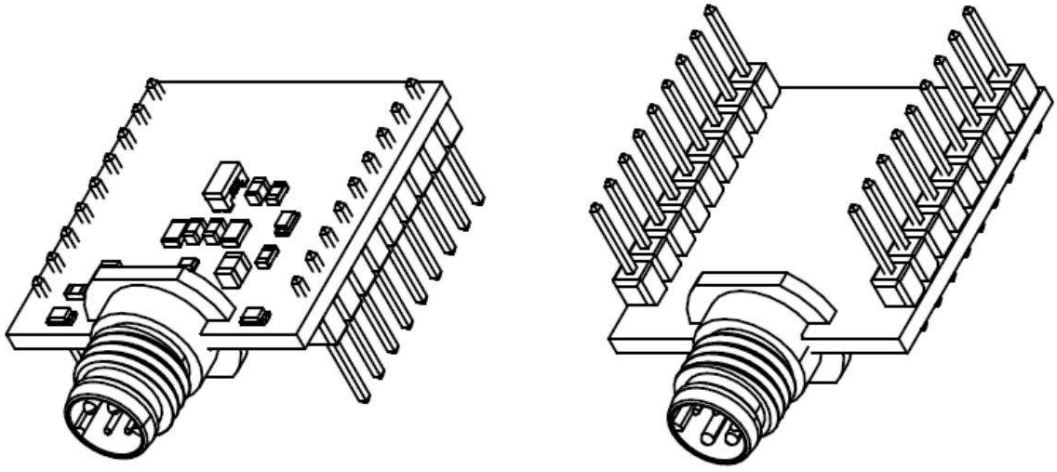


Figure 6: M8 Hybrid Male daughterboard.

The pinout of the network communication daughterboard is shown below in Figure 7.

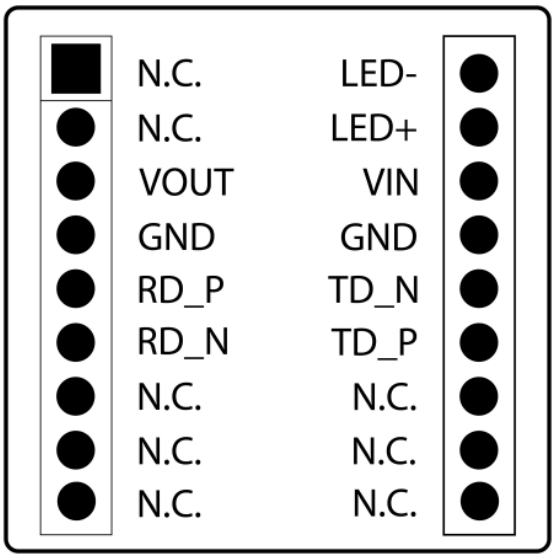


Figure 7: Network communication daughterboard pinout.

As can be seen, the daughterboard interface uses the following signals:

- Differential pair TD (`TD_N` and `TD_P`) for data transmission/reception in 100BASE-T1. With 100BASE-TX. This pair is only used for data transmission.
- 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 a positive terminal of the external power supply that is supplied to the development board via the connector (e.g. daughterboard).
- `VOUT` is a positive power supply terminal that is supplied from the development board to the daughterboard connector. On the development board, this signal is directly connected with the main power supply for the *periCORE*.
- `N.C.` means not connected. The function of these pins is not defined at the moment and they are saved for future use.

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

2.3 Sensor/actuator Interface

The development board has a slot for placing sensor/actuator daughterboards. The footprint for the sensor/actuator daughterboards is compatible with mikroBUS standard. However, there are two additional signals to the footprint: `24V` and `AN2` (second analog input channel). The dimensions and the pinout of the sensor/actuator daughterboard slot on the development board is shown below:

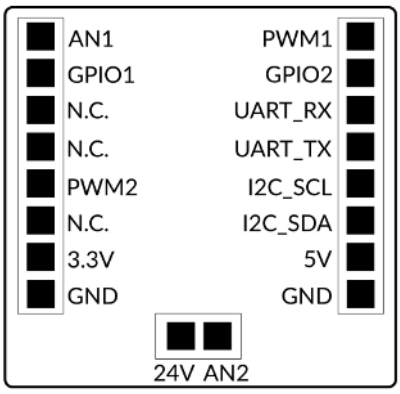


Figure 8: Sensor/actuator daughterboard pinout.

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 5 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 UART interface

The UART interface (`J16`, see again also Figure 5 number 5) of the *periCORE* module can be accessed by e.g. using the provided FTDI TTL232R-3V3 device connecting it via USB to the *periMICA* device. For more information also refer to section 3.5.

3 System Architecture

3.1 Overview

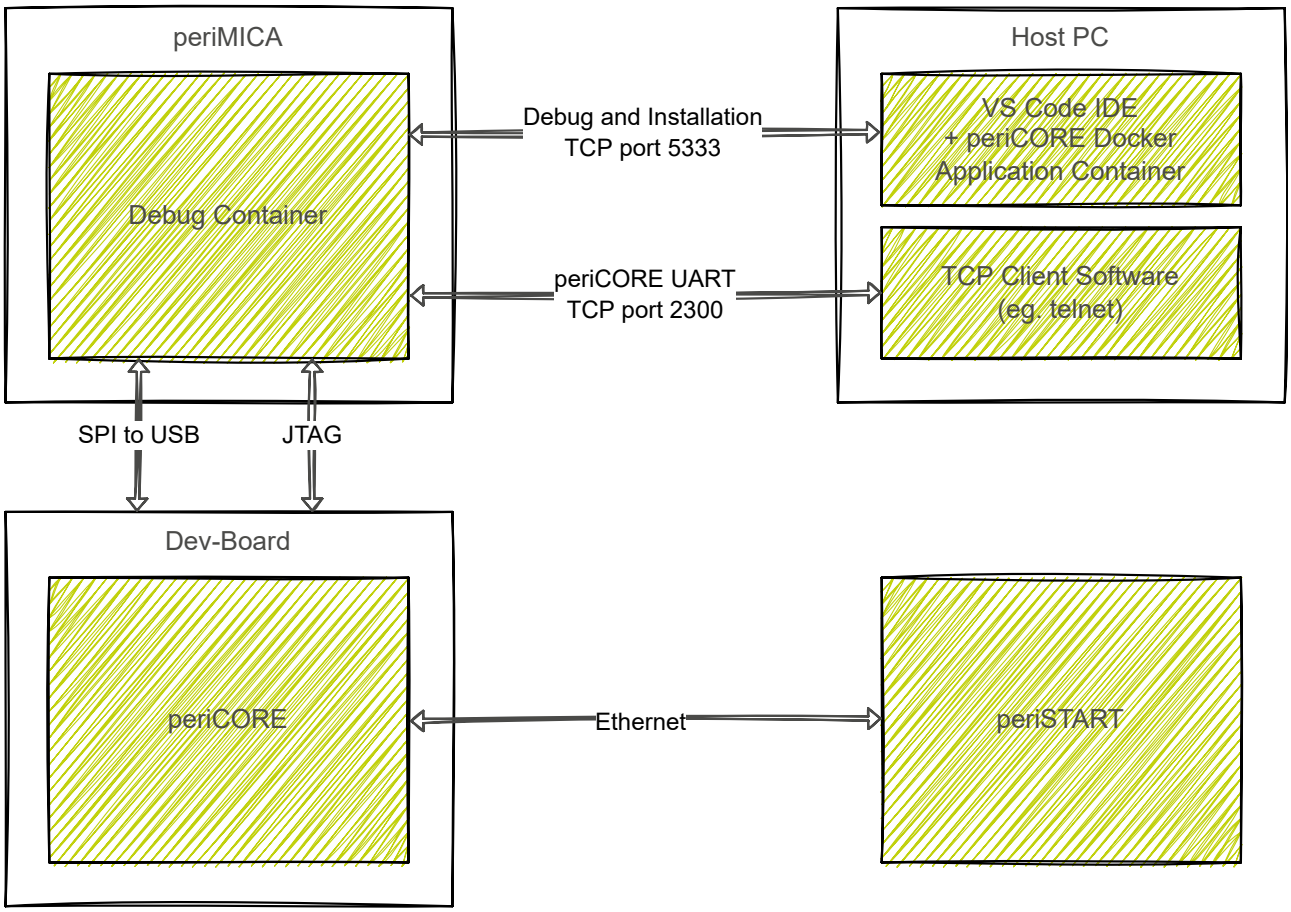


Figure 9: System architecture of *periCORE development kit* setup.

The basic components for developing customized *periCORE* applications require a PC, a *periMICA* as debugging interface and the development board itself with the *periCORE* device as heart of it (Figure 9).

As building and debugging environment serves Visual Studio Code © with the *Remote Container Extension* accessing a docker container that is provided by *Perinet* (Figure 10).

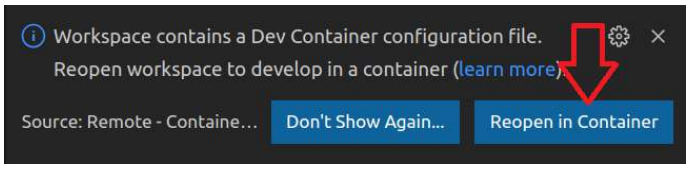


Figure 10: Using Remote Containers Extension in Visual Studio Code ©.

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 guide lines from periCORE Development Kit Setup Application Note [6].

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>/pericore.application.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

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

Therefore a *periSTART* device is used. After connecting the development board to the *periSTART* and attaching it to a network switch it should be available via network. The name printed on the label (Figure 11) can be used to validate network functionality is working correctly.

Figure 11: periSTART device with label information.

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

3.4 Debug interface

After installing the *pericoredbg Container* to the provided *periMICA* [6] 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* is running an *JLinkGDBServerCLExe* instance triggered by a socat process providing a TCP connection via port *5333*.

Just like for all *Perinet Smart Compoments* mDNS names can be used to access the *periMICA* containers via network. Hence Visual Studio Code © debug settings 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`. Therefore modify the `DEBUG_TARGET`, see also listing 2.

```
"DEBUG_TARGET":"pericoredbg-periMICA-serno.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-serno.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 there is any need to adapt the TCP port or some other J-Link configuration is wanted this can be done by using SSH into the *pericoredbg Container*, via e.g. using Putty (the mDNS name of the container can be used to access it).

For SSH access an One time password is necessary which needs to be generated from the *pericoredbg Container* webpage, that again can be accessed via mDNS url or from the *periMICA* home page by clicking on the *pericoredbg Container* icon (Figure 12, also refer to periMICA User Guide [8] for more information).

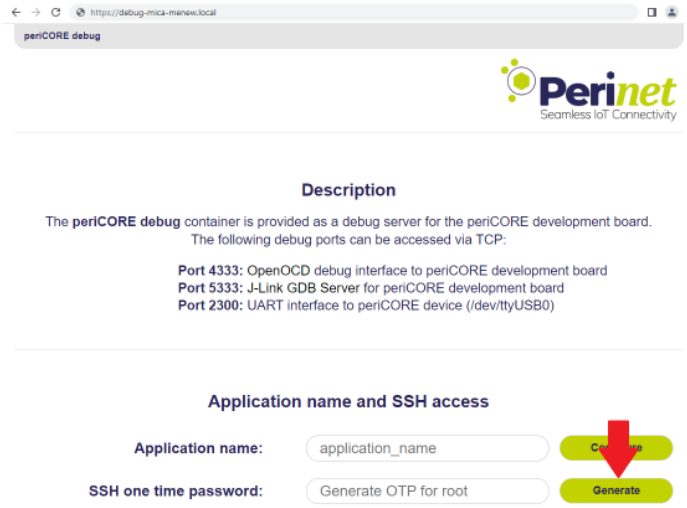


Figure 12: Excerpt from *pericoredbg Container* Web UI, using mDNS in the browser. Generate SSH One time password for user `root` access by clicking on the *Generate* button.

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

3.5 UART interface

Just like it is done for the debugging interface access to UART of the periCORE can be used for debugging via *pericoredbg Container* TCP port *2300*. Again Putty could be used to access

this interface. On *Linux* based OS telnet might be a more common tool, however.

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 periCORE Firmware Development Application Note [7] for more information on this)

4 Software Development

4.1 periCORE Software API structure

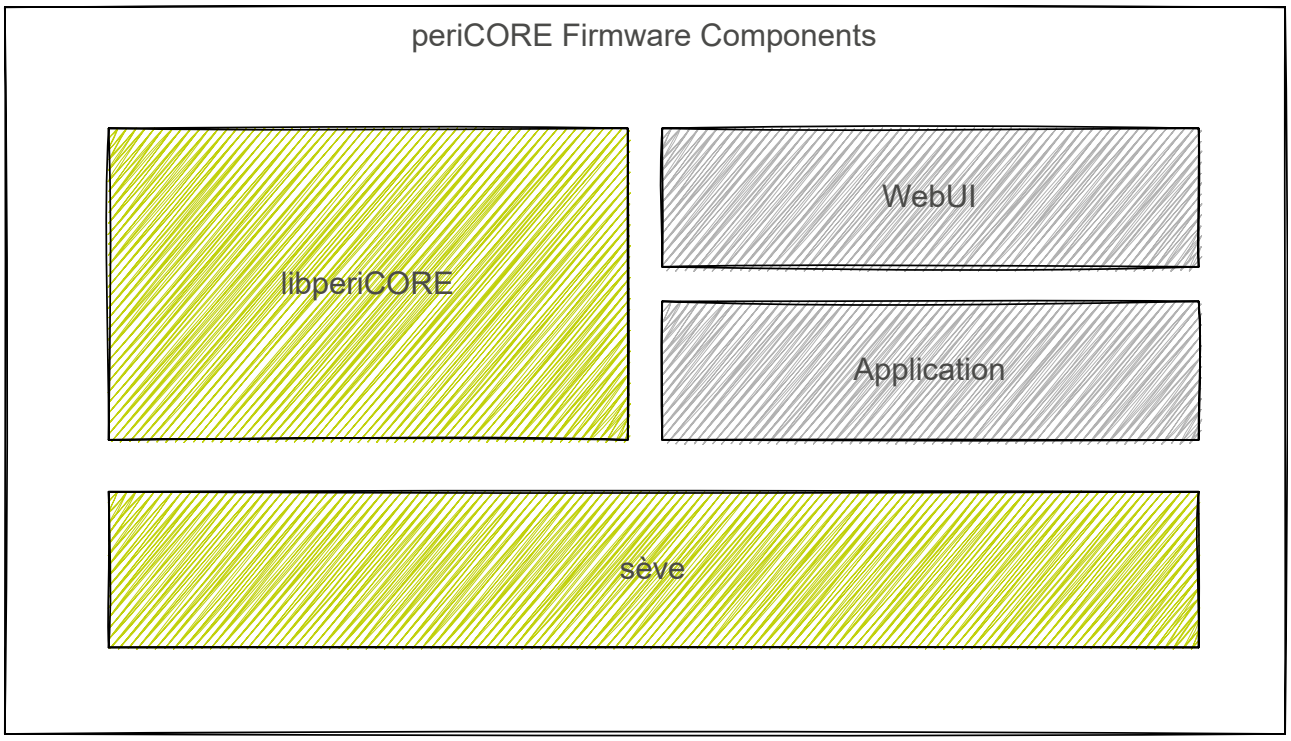


Figure 13: Firmware Architecture of *periCORE* based applications. Green boxes indicate components that are common for all applications. On the other hand, grey boxes define application specific components.

The basic idea of the *periCORE development kit* from software perspective is to customize applications on top of the Operating system (*sève*) and the so called *libperiCORE*. The application part is implemented by mainly extending the functionality given through *libperiCORE* in detail by creating additional endpoints for both MQTT Client and HTTP Server.

The Web UI can also be customized on top of a minimized *Javascript* Framework implemented.

4.2 sève OS

Section 4.3 requires basic knowledge of *sève* regarding how data is represented by the *Buffer* data type (`seve::lib::memory::Buffer`) and also how outports and inports function between different components, namely focus on:

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

Therefore refer to sève Operating System Datasheet [9].

4.3 Extend libperiCORE Network Services

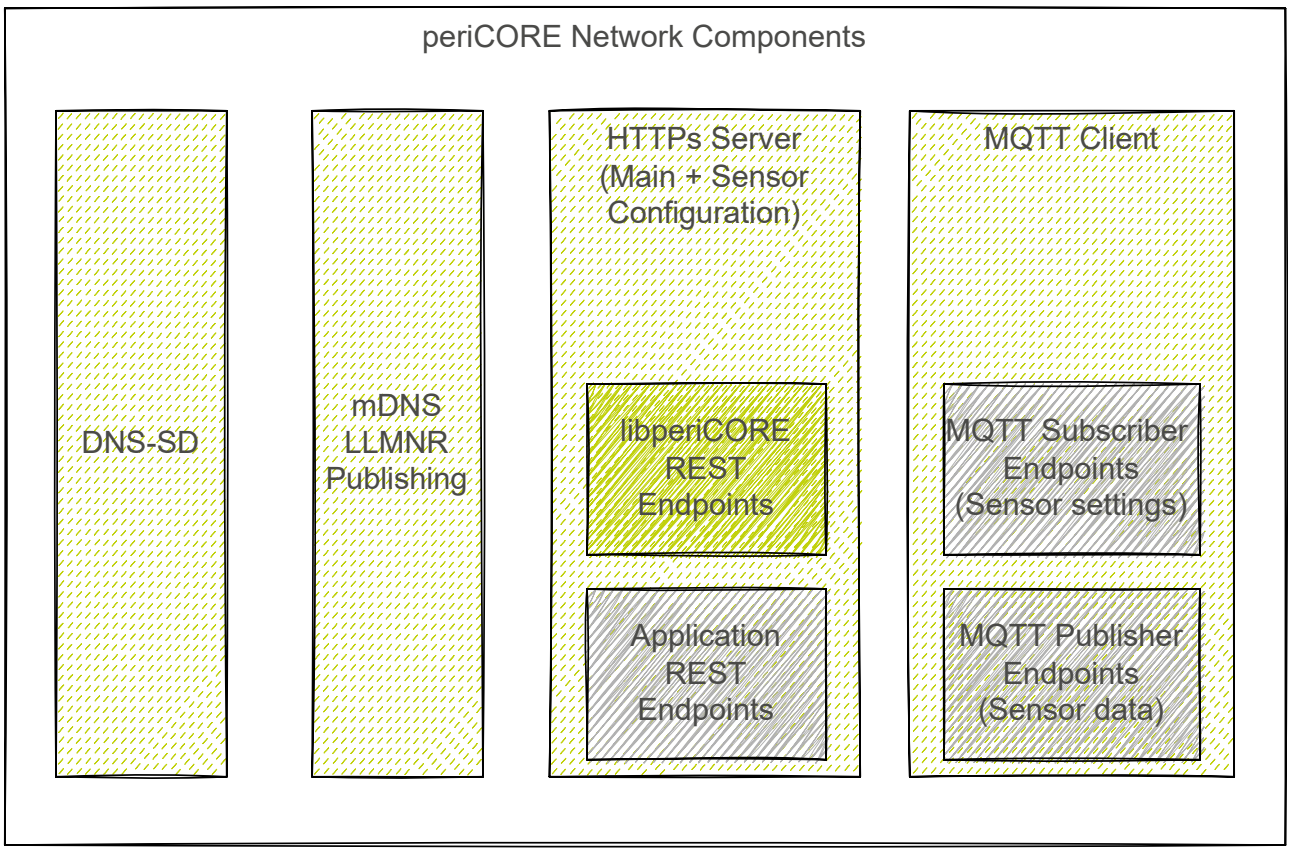


Figure 14: Network Architecture of a *periCORE* based firmware application. Green and green-dashed boxes indicate components that are provided by *libperiCORE*, hence are common for all applications. On the other hand grey boxes define application specific components, like Endpoints used for both HTTP Server and MQTT Client.

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

Due to a less configuration approach only IPv6 link local addresses are being used for network communication through all *periCORE* based devices which has some implications on the usage, however most distributions and browsers support IPv6 link local addresses. In case you are encountering problems anyways, refer to section 5.

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

- ***HTTP Server:*** main interface with REST-like API for the user to configure a *Perinet Smart Components* device or perform firmware updates. The HTTP interface can also be accessed by a provided Web UI which is mainly general for most applications, but allows customization as well (See section 4.4).

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

Since security becomes more and more important in networking, the HTTP Server provides settings to use mTLS and trusted certificates, which restricts access to the HTTP Server only to users with valid client certificates. Also see periCORE Datasheet [5] for more information.

- ***mDNS/LLMNR Publishing:*** Since IPv6 addresses can be very cryptical, LLMNR and mDNS hostname publishing are being used to allow access of *Perinet Smart Components* by a unique device name, without the need of having to type IP addresses.
- ***DNS-SD:*** DNS Service discovery is used to browse for services of a specific type. On one hand *periCORE* based applications use it to announce meta data about its given services (like *port*, *url* and *application* information). On the other hand *periNODE* devices make use of it to discover the hostnames of MQTT Brokers in the network that can be used to configure the broker of the *libperiCORE* MQTT client.
- ***MQTT Client:*** A basic MQTT Client implementation is given by *libperiCORE*, also supporting TLS / mTLS features similar to the HTTP Server. Functionality can be implemented by creating MQTT Client Endpoints for both publishing sensor data or subscribing to events using this to 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 like *GET*, *PATCH*, *PUT*, *POST* and *DELETE* being called automatically by the HTTP Server, when the according method and URI / route is being

requested (listing 5). That is why the *Endpoint* object also needs to have a unique URI to be identified which must be set by 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.

For *PATCH* and *PUT* methods default implementations are given already. Both callbacks proxy the HTTP request payload as a *Buffer* to an external component. Thus in these cases according 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 basically caches the latest *Buffer* (which is mostly a JSON message) provided from an external component, and sends it as response to the HTTP client when receiving a HTTP *GET* request under the corresponding URI (See also figure 15).

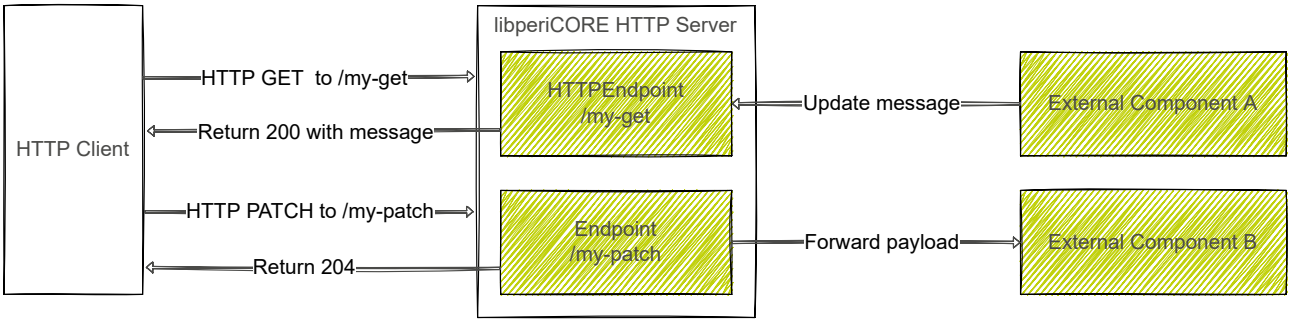


Figure 15: Example application HTTP Server with two endpoints, one for either HTTP GET and PATCH request handling.

As data representation basically JSON is used for HTTP request and response communication.

In case mTLS is active in the application the `handle_request` method of the *Endpoint* class determines which user role is needed for the according URI and HTTP method combination. Currently the following user roles are implemented by default (listing 8):

- *PATCH* / *PUT* requests: basically requires *ADMIN* role, only for URI that start with */config* or */sample* require *SUPER* rights.
- *GET* / *DELETE* / *POST* requests: default user role is set to *READER* which is intended for *GET* requests but only also given for *DELETE* and *POST* since these HTTP methods have not been used / implemented, yet.

```
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.

Thereby the three different user role types for *periCORE* based applications are defined as following:

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

Also see periCORE Firmware Development Application Note [7] 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 by either publishing a message when a *MQTTPublisherEndpoint* triggers its outport (e.g. by a timing event) or forwards a received MQTT message to the *MQTTSubscriberEndpoint* that obeys an according MQTT topic.

Both MQTT endpoint classes need to be defined by a MQTT topic upon 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 directly added to the *libperiCORE MQTT Client*, instead the outport of the endpoint is connected with the publish inport of the *libperiCORE MQTT Client* (listing 10). So the endpoint has indirect control triggering the publishing when a new message from the connected external component was received (See figure 16).

```
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.

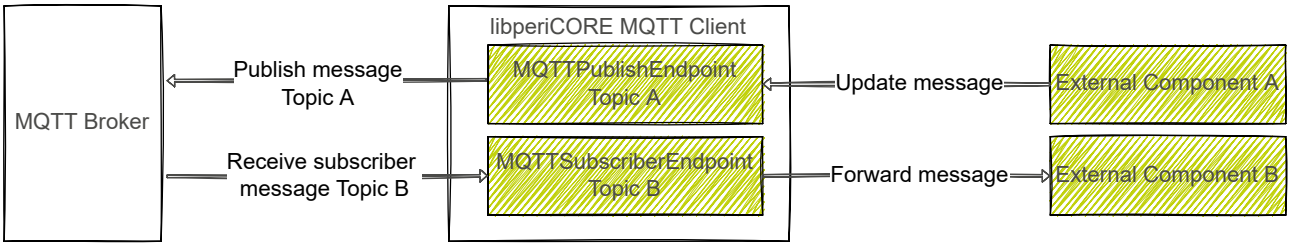


Figure 16: Example application MQTT Client with one *MQTTPublisherEndpoint* and one *MQTTSubscriberEndpoint*.

- *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.

Just like for the HTTP Server JSON is used as data representation format for MQTT data communication.

Also see periCORE Firmware Development Application Note [7] 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 to a firmware update image. The Web UI can not be installed or updated using the debugging tools, instead the Firmware update function via Web UI or REST API call of the *periCORE* device needs to be used.

The sources of the Web UI binary are basically comprised of js and html files. Most of the functionality is done by using *Javascript* to have a Web application like feeling meaning for example that only specific page content is loaded when navigating in between different Web UI tabs. The header, footer and also basic css and js are loaded only once, just the actual content in between is reloaded on demand. Another benefit of this is that it keeps the request load low on the *periCORE* device, that has a limited performance especially when it comes to HTTP using TLS features.

Also due to performance issues the Web UI Makefile decodes css and png file links into Base64 representative strings of the according files replacing them in the according Javascript and html files. It is recommended to use png files as picture file format, since only a total Web UI binary image size of *256KB* is allowed to build.

Again see periCORE Firmware Development Application Note [7] for an in depth example on how to develop customized Web applications for *periCORE* devices.

4.4.2 TLS session resumption

Another way of increasing Web UI performance is to make use of TLS session resumption. A usual HTTP call to a *periCORE* based device with TLS enabled takes about *700 ms*. Having mTLS enabled it takes even longer with about *1300 ms* for each request. However when using TLS session resumption the TLS handshake will be only performed on the first request, after that this initial TLS session is used for following requests by both client and server. Which means that these requests then only need *100ms* to *200ms* each regardless of using mTLS or not.

The *periCORE* HTTP Server backend supports TLS session resuming, however from client side some browsers have it disabled. Newer Firefox versions seem to not use it (anymore), while Microsoft Edge and Google Chrome are supporting it as of yet.

4.4.3 Html Dynamic Content Modification

Like mentioned above the content of the Web UI can be divided into header, content and footer. The header and footer thereby remain constant, only the content in the center is dynamically changed (See figure 17).

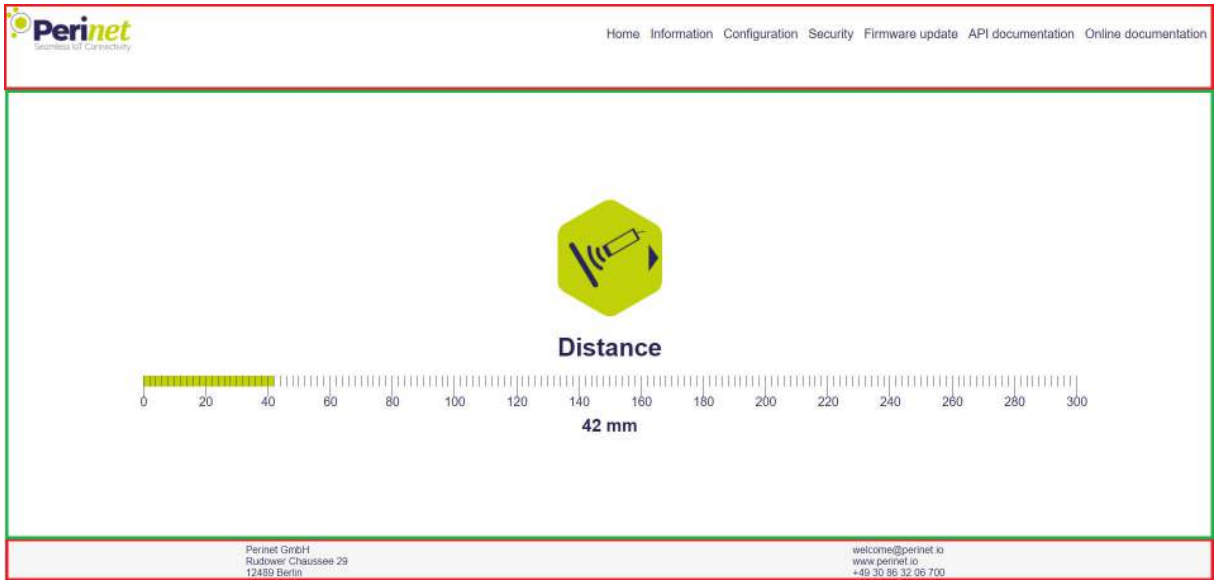


Figure 17: Web UI structure of *periCORE* based applications. Red boxes indicate static content (header and footer), whereas the actual content (green box) is dynamically modified when navigating to other web pages using the header tabs.

In order to allow content being loaded automatically means, no html files need to be modified, instead html DOM elements are created by Javascript functions. For example when looking at the content function to create the DOM elements for the Firwmare Update web page (Listing 12 taken from application repository path `webui/fs_periNODE_distance/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:2px" 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_periNODE_distance/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.

Therefore the `update_content` function needs to be called by passing a callback function (`reload_update` in this case) which in turn will be called when the user enters the according

web page. The first parameter of the `update_content` is an identifier for the corresponding web page, which is used by the framework (`peri_base.js`) to identify the callback (See also calls of `reload` functions from 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_periNODE_distance/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

In order to add a new page to the Web UI follow these steps:

- Copy the `home.html` from `webui/fs_periNODE_distance` folder into the same destination and name it accordingly, e.g. `custom.html`.
- In the `webui/fs_periNODE_distance/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_periNODE_distance/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` needs to be called once for the case that the js file is loaded initially for the first time.

- now the `reload_custom` callback needs to be registered, hence added in the `peri_base.js` file like following:

```
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.

It is important to name the function like `load_<uri>`. The URI needs to be the same as the value passed to the `reload` function for the navigation link (see again listing 15). The `update_content` function expects as first parameter the URI and as second parameter the callback defined in the according js file, just like it is called in the js file.

Note: You might be wondering why the `update_content` function is only loaded in case the javascript file was already loaded. This is due to the fact, that the javascript file with the actual reload callback is loaded asynchronously. So in the example of the `load_custom` function which is calling the `load_js_script` function it is not guaranteed when the latter function returns, that the javascript file was loaded already. Therefore the loaded javascript file itself is calling the `update_content` function for one time.

4.4.6 Loading css files

Additional Stylesheet files can also be added by calling the according Javascript function `load_css_script` (See listing 18 taken from `webui/fs_periNODE_distance/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_periNODE_distance/js/peri_base.js` by loading your customized style sheets instead.

It is important to understand, that the Web UI Makefile is replacing the css links given in all js and html by its Base64 string representation (compare listings 18 and 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

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

Note: Also similar to css files the Makefile does not add the actual image files to the Web UI binary. When other file formats like jpeg need to be supported, we recommend to use the Base64 string representation as well. Therefore changes to the Web UI Makefile would be necessary.

4.4.8 Debugging Web UI changes

In order to test a new implemented Web UI you need to create the firmware update image and update the *periCORE* from the development board via REST API or Web UI. Using the debugger via *pericoredbg Container* can not be done since the Web UI files are not reloaded into the flash memory. See also periCORE Firmware Development Application Note [7] for more information.

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

5 Troubleshooting

I can not reach the webpage of the periMICA *device.*

Make sure that your OS is supporting mDNS resolution and IPv6 link local addresses. Though the *periMICA* device supports IPv4 as well the *periMICA* containers and *Perinet Smart Components* can only be accessed using IPv6 link local. Check that in a command terminal (Under *Windows* press the **Windows** button and enter `cmd`) `ping` is working 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 can not access IPv6 link local addresses with certain browsers

This should mainly affect you if your host OS is Debian / Ubuntu based. Since most browsers struggle with using IPv6 Link local address zone identifiers under those Operating Systems. Firefox might be an exception here but it is not recommended since the Firmware Update from Firefox is not working for *Perinet Smart Components*. There are basically 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 [8] 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 comes without any labeling on itself. For interpretation of the label of periCORE module, please refer to the periCORE datasheet.

Ordering

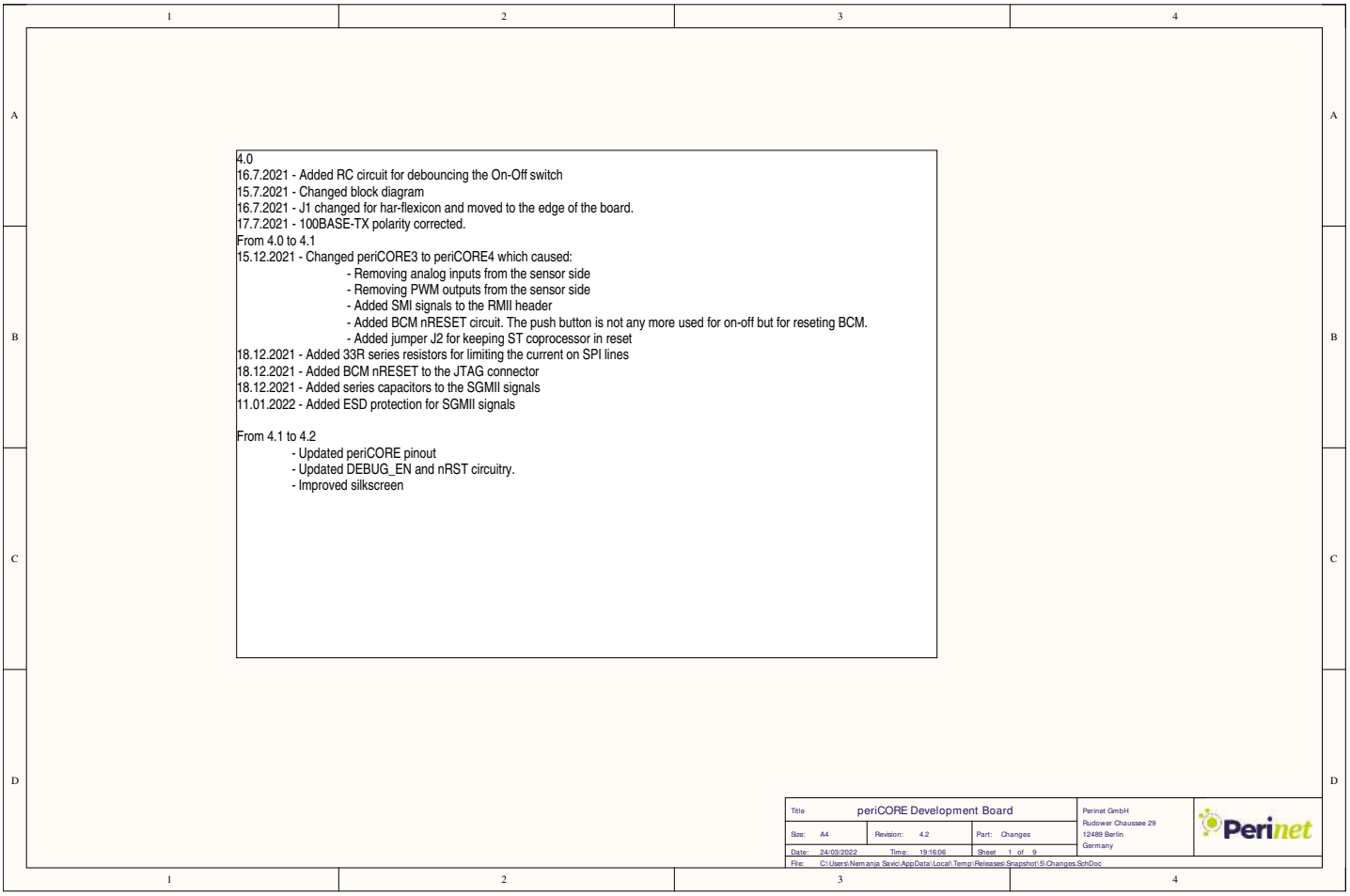
For ordering periDEVboard 3.3 please contact *sales@perinet.io*.

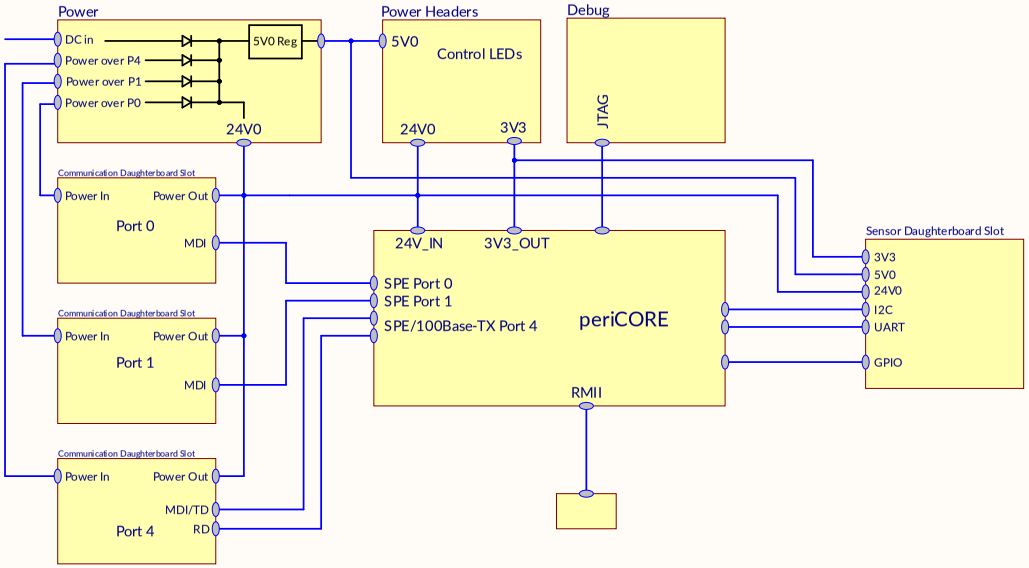
7 Contact & Support

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`

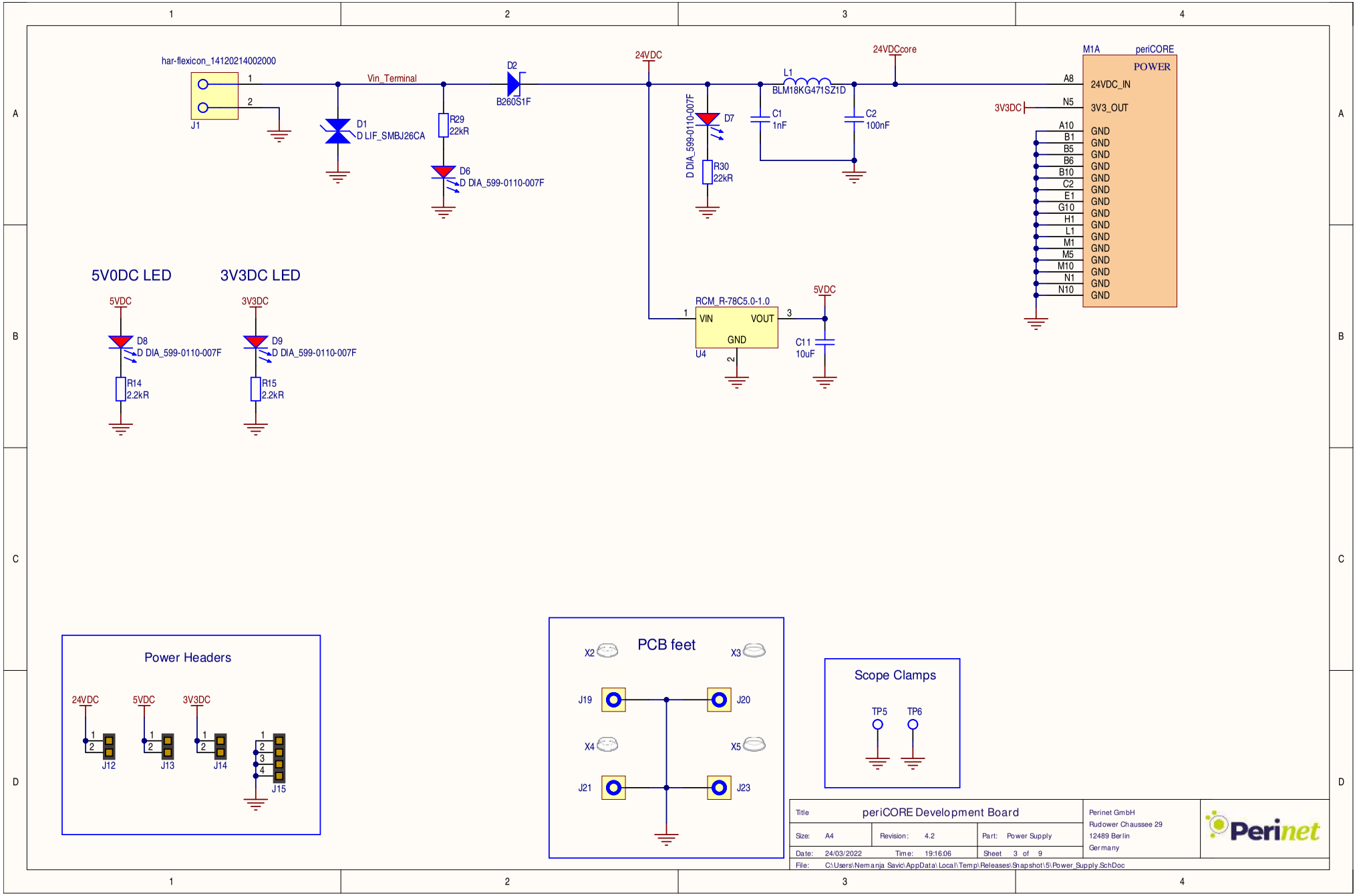
A Development Board

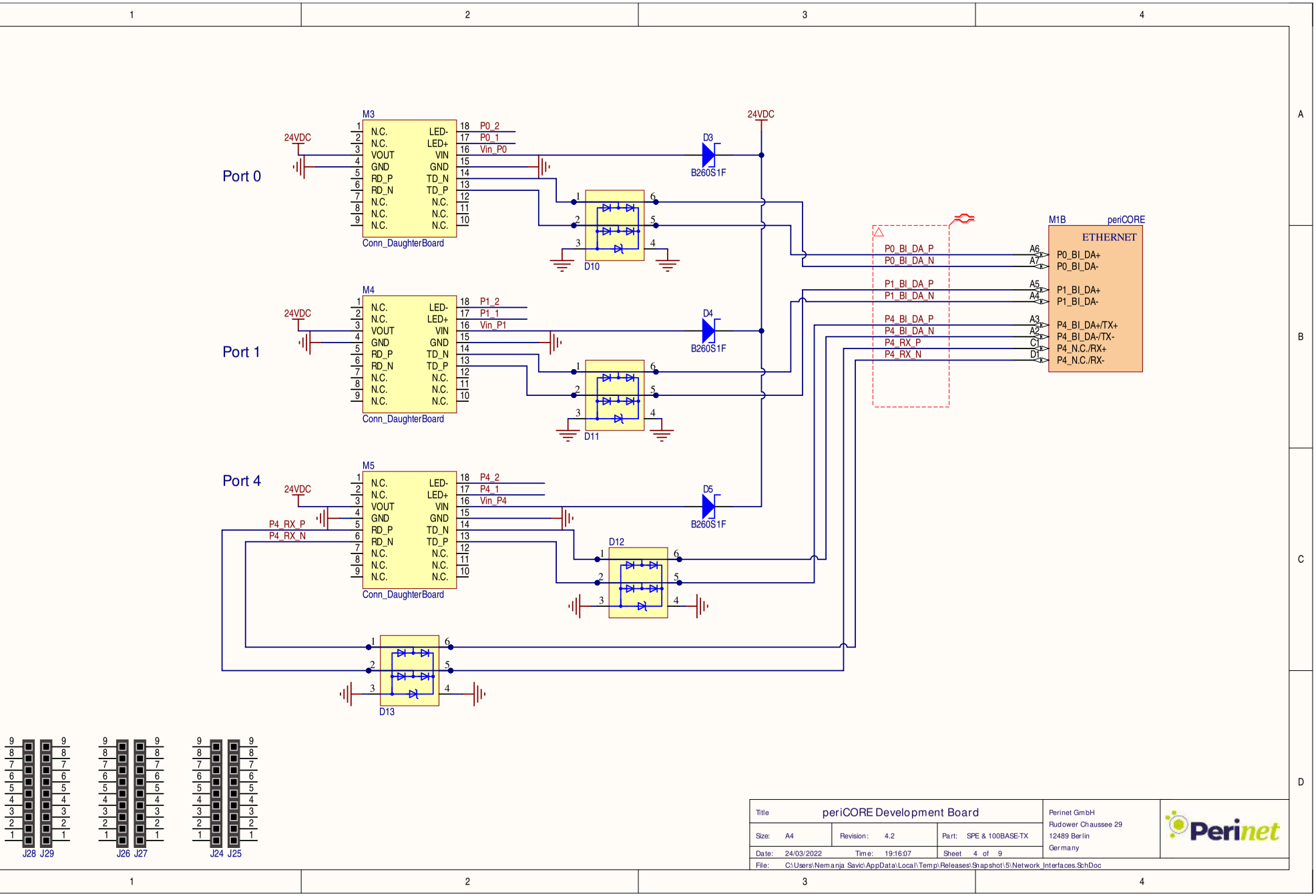




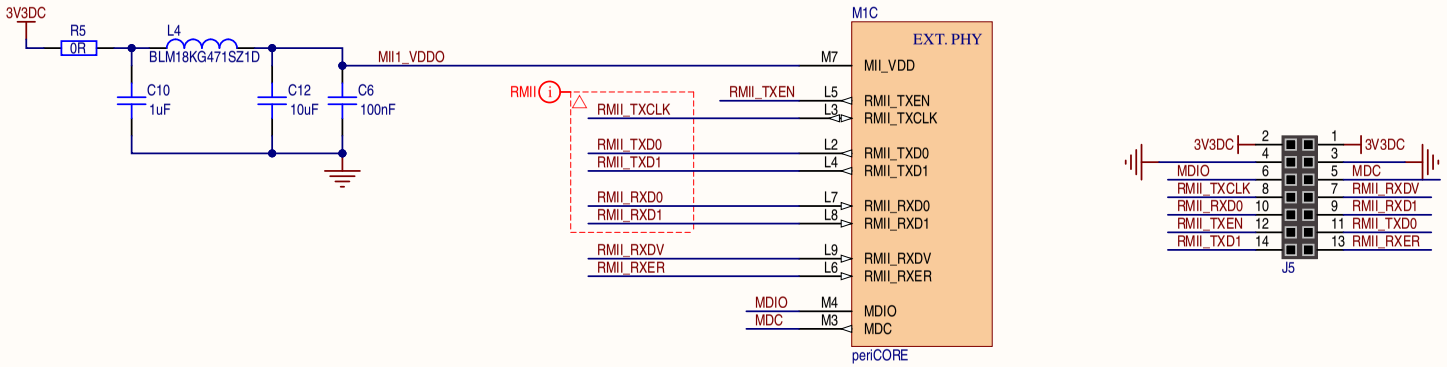
Logo Perinet Logo Perinet

Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: Overview	
Date: 24/03/2022	Time: 19:16:06	Sheet 2 of 9	
File: C:\Users\Nemanja.Savic\AppData\Local\Temp\Releases\Snapshot\5\Overview.SchDoc			

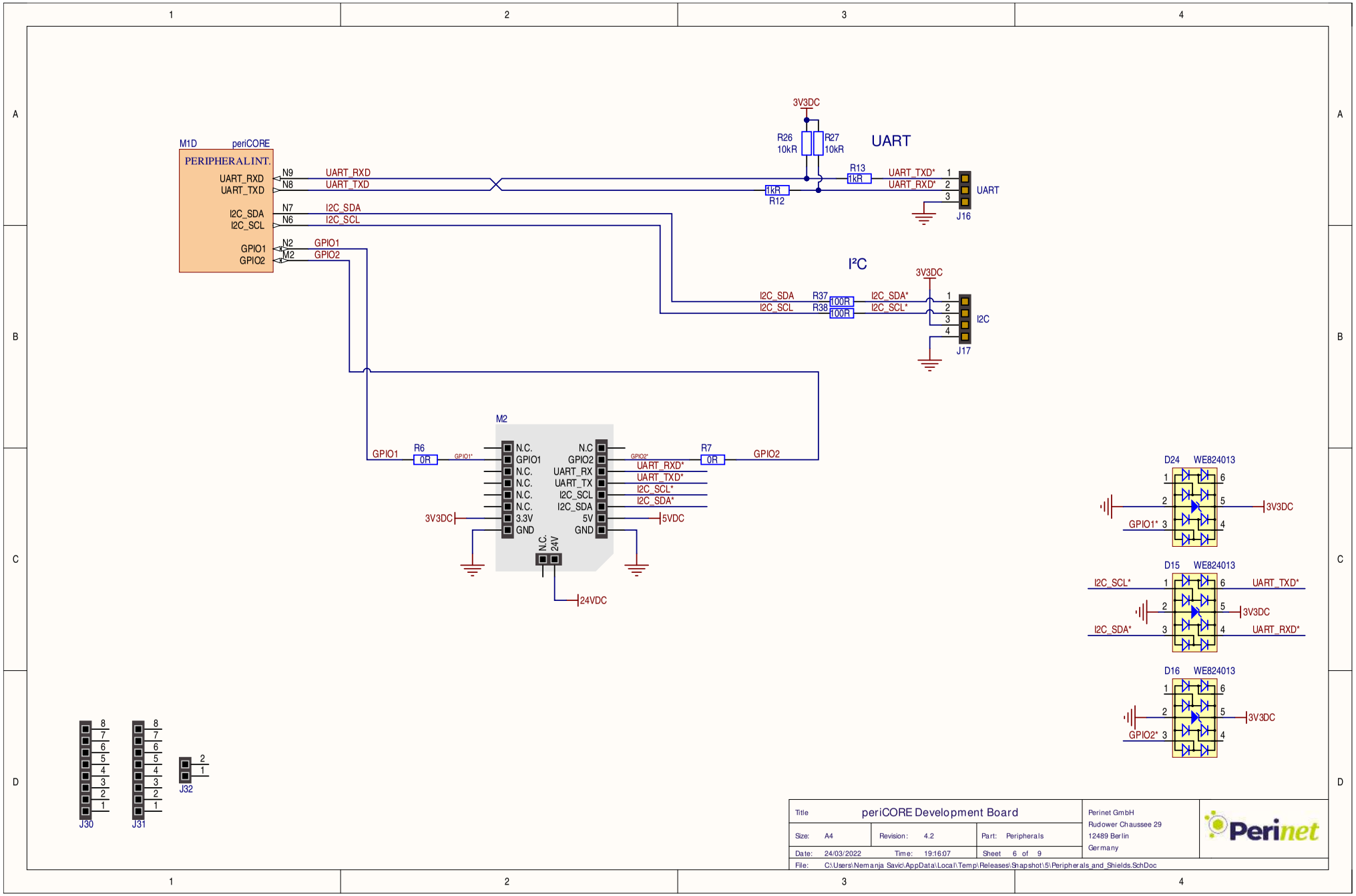

har-flexicon_14120214002000
J1
Vin_Terminal
D1
D LIF_SMBJ26CA
D2
B260S1F
R29
22kR
D6
D DIA_599-0110-007F
24VDC
D7
D DIA_599-0110-007F
R30
22kR
L1
BLM18KG471SZ1D
C1
1nF
C2
100nF
24VDCcore
M1A
periCORE
POWER
A8 24VDC_IN
N5 3V3_OUT
3V3DC
A10 GND
B1 GND
B5 GND
B6 GND
B10 GND
C2 GND
E1 GND
G10 GND
H1 GND
L1 GND
M1 GND
M5 GND
M10 GND
N1 GND
N10 GND
5V0DC LED
3V3DC LED
5VDC
D8
D DIA_599-0110-007F
R14
2.2kR
3V3DC
D9
D DIA_599-0110-007F
R15
2.2kR
RCM_R-78C5.0-1.0
VIN
VOUT
GND
U4
C11
10uF
5VDC
Power Headers
24VDC
5VDC
3V3DC
J12
J13
J14
J15
PCB feet
X2
X3
J19
J20
X4
X5
J21
J23
Scope Clamps
TP5
TP6
Title
periCORE Development Board
Size: A4
Revision: 4.2
Part: Power Supply
Date: 24/03/2022
Time: 19:16:06
Sheet 3 of 9
File: C:\Users\Nemanja.Savic\AppData\Local\Temp\Releases\Snapshot\5\Power_Supply.SchDoc
Perinet GmbH
Rudower Chaussee 29
12489 Berlin
Germany
Perinet



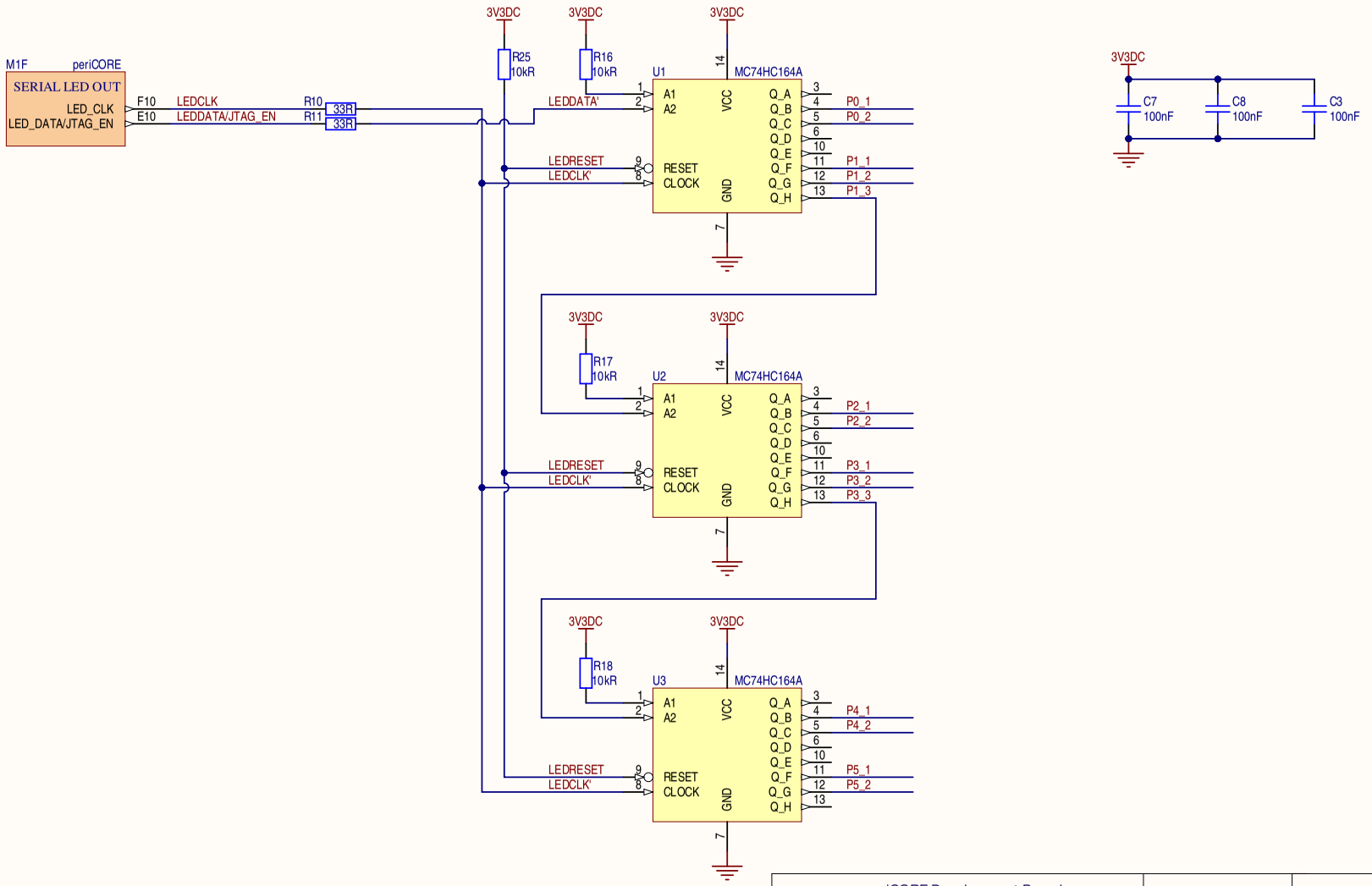
Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: SPE & 100BASE-TX	
Date: 24/03/2022	Time: 19:16:07	Sheet 4 of 9	
File: C:\Users\Nemanja Savic\AppData\Local\Temp\Releases\Snapshot\5\Network_Interfaces.SchDoc			



Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: RMII	
Date: 24/03/2022	Time: 19:16:07	Sheet 5 of 9	
File: C:\Users\Nemanja.Savic\AppData\Local\Temp\Releases\Snapshot\5\RMII.SchDoc			

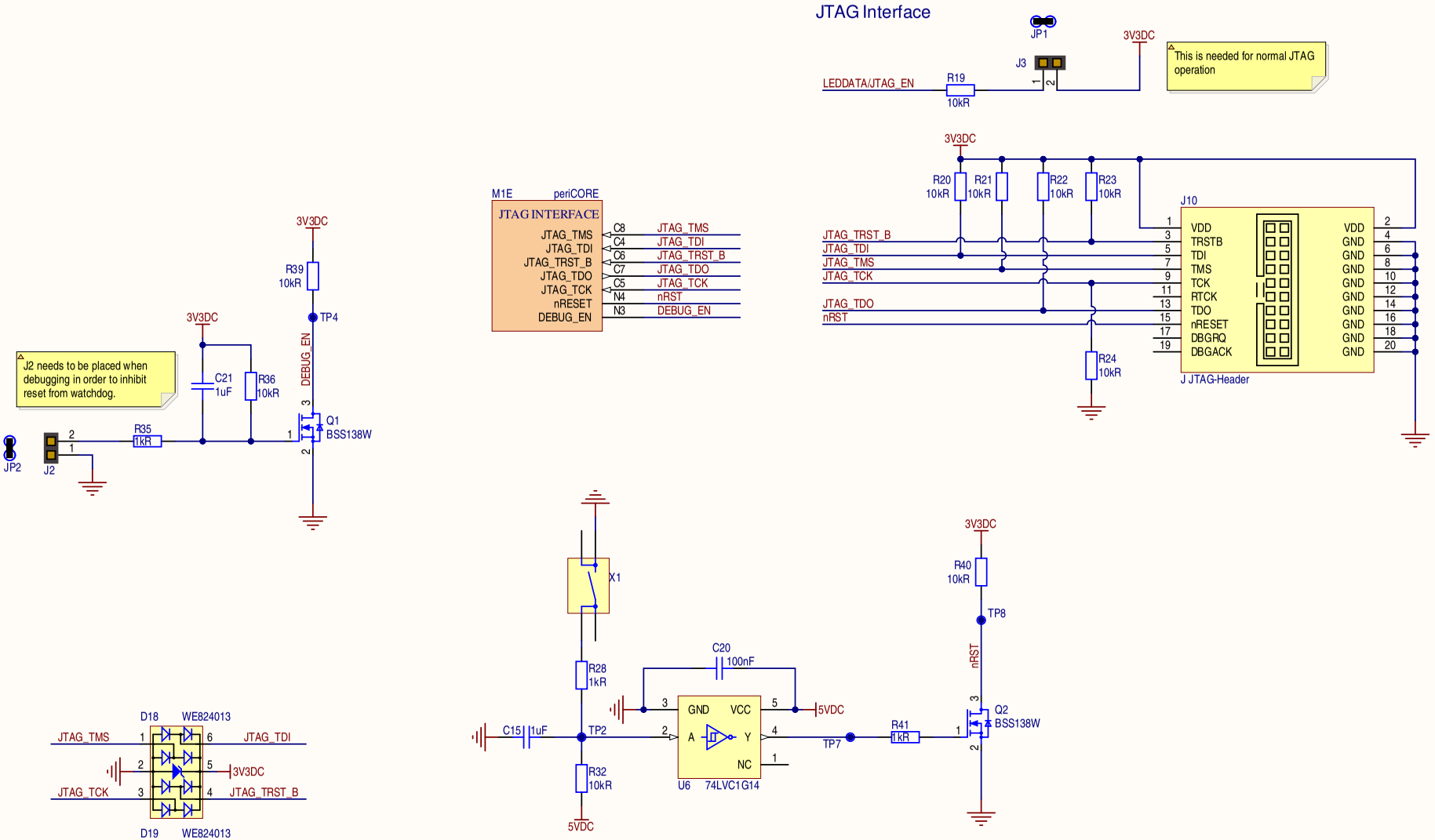


Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: Peripherals	
Date: 24/03/2022	Time: 19:16:07	Sheet 6 of 9	
File: C:\Users\Nemanja.Savic\AppData\Local\Temp\Releases\Snapshot\5\Peripherals_and_Shields.SchDoc			



Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: Link Status LEDs	
Date: 24/03/2022	Time: 19:16:07	Sheet 7 of 9	
File: C:\Users\Nemanja Savic\AppData\Local\Temp\Releases\Snapshot\5\Link Status LEDs.SchDoc			

JTAG Interface



Title	periCORE Development Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 4.2	Part: JTAG	
Date: 24/03/2022	Time: 19:16:07	Sheet 8 of 9	
File: C:\Users\Nemanja.Savic\AppData\Local\Temp\Releases\Snapshot\5\JTAG.SchDoc			

B Network Daughterboards

B.1 M8 Hybrid Male Connector Daughterboard

The M8H Male Daughterboard is a network daughterboard which allows using the M8 Hybrid Male connector with the *periCORE development board* (see figures 18 and 19).

Figure 18: M8H Male Daughterboard.

WIRE COLOURS AND PIN ASSIGNMENT

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

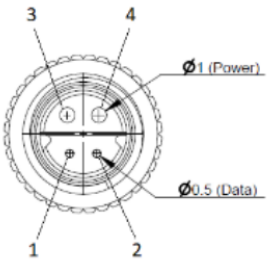


Figure 19: M8H Male connector drawing and pinout.

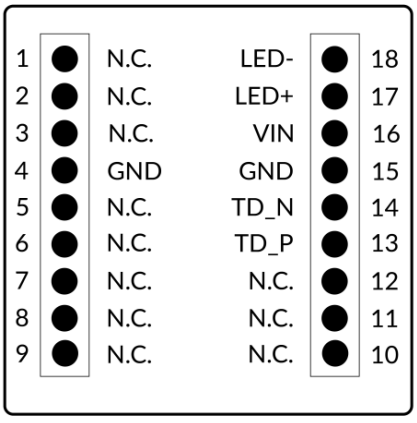
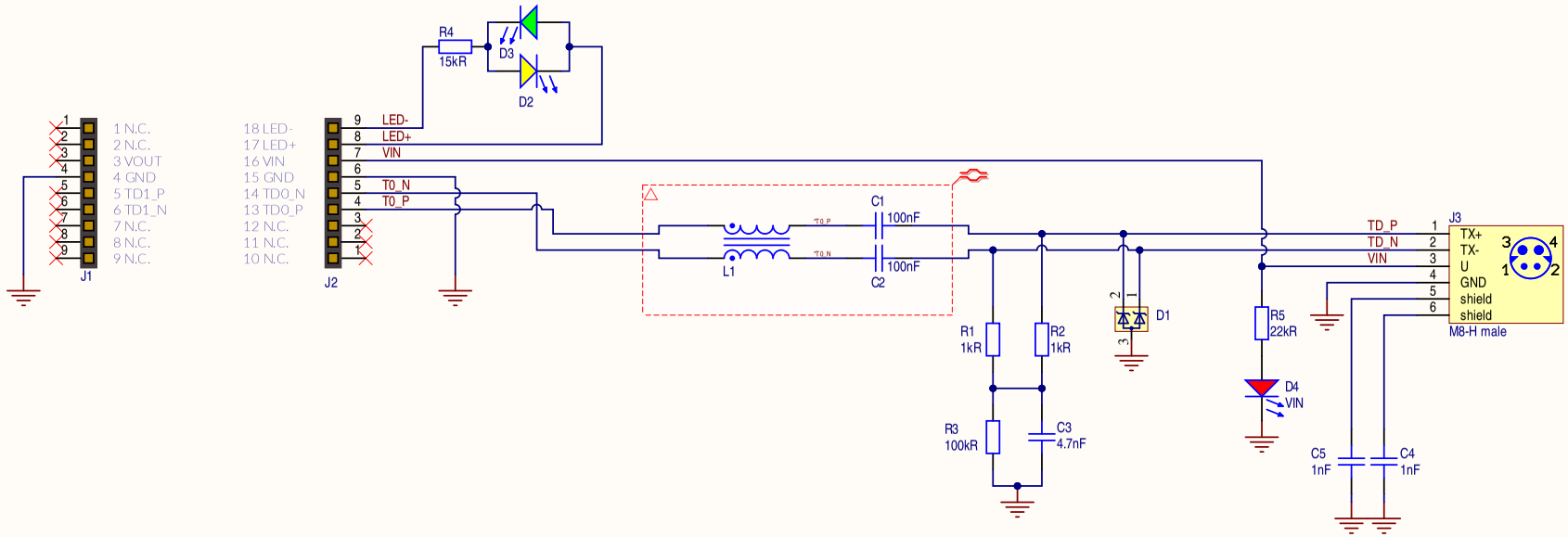


Figure 20: M8H Male daughterboard pinout.

- 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.



Title	M8H Male Daughter Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.0	Part: Overview	
Date: 22/07/2021	Time: 16:23:15	Sheet 1 of 1	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\M8HM_Daughterboard\M8HM_Daughterboard.SchDoc			

B.2 M8 Hybrid Female Connector Daughterboard

The M8H Female Daughterboard is a network daughterboard which allows using the M8 Hybrid Female connector with the *periCORE development board* (see figure 21).

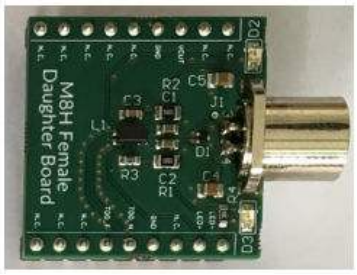


Figure 21: M8H Male Daughterboard.

WIRE COLOURS AND PIN ASSIGNMENT

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

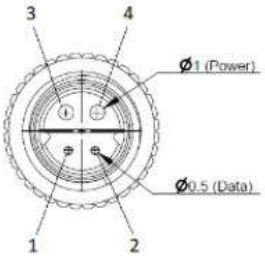


Figure 22: M8H Female connector drawing and pinout.

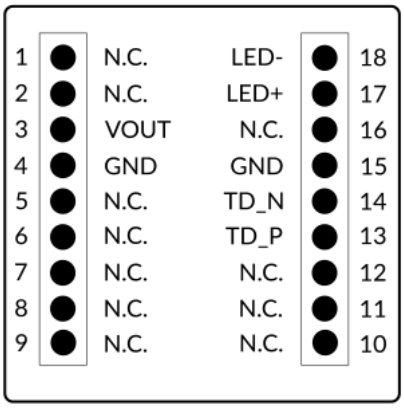
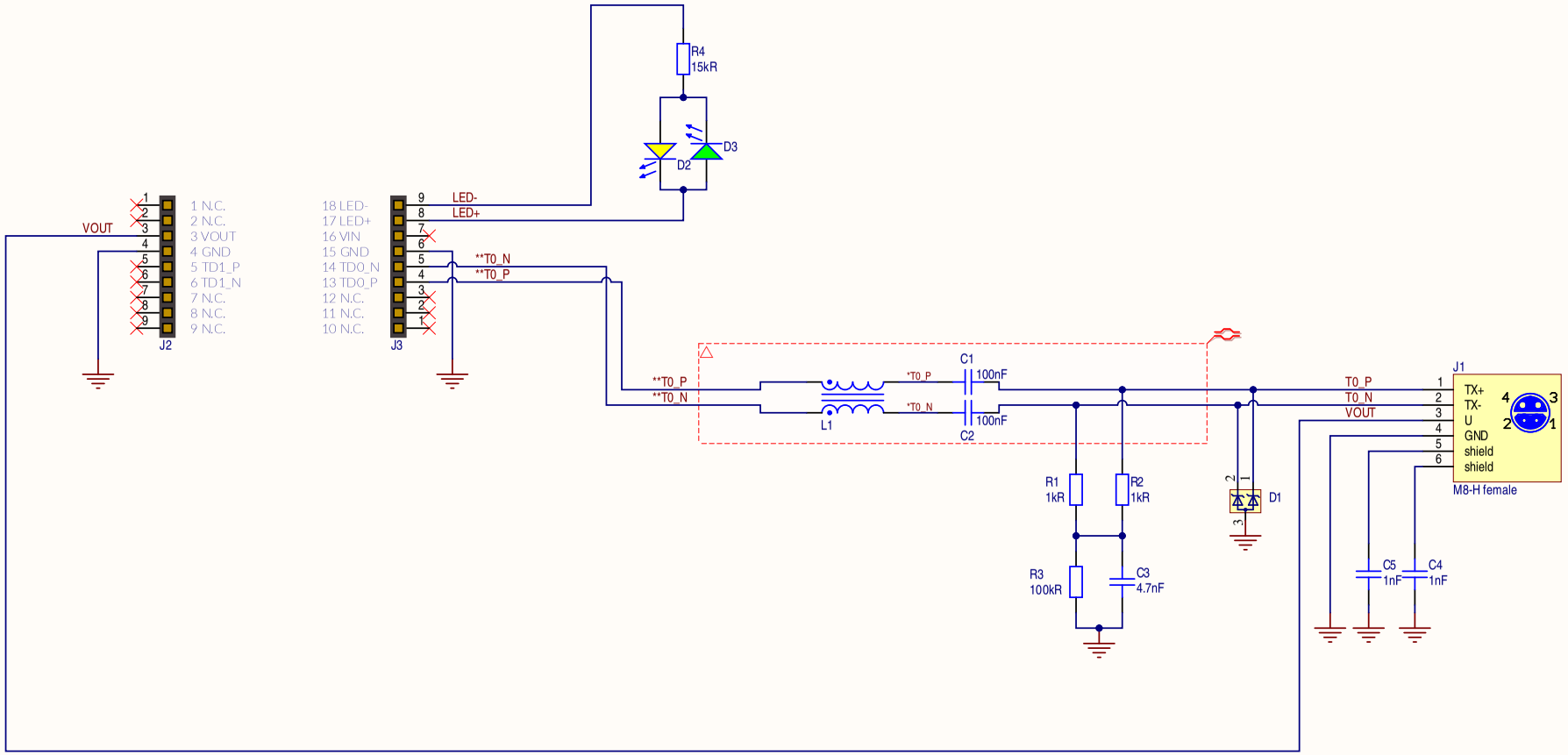


Figure 23: M8H Female daughterboard pinout.

The pinout of the M8H Female Daughterboard (see figure 23) is:

- 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.



Title	M8H Female Daughter Board		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.0	Part:	
Date: 23/07/2021	Time: 09:38:44	Sheet 1 of 1	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\M8HF_DaughterBoard\M8HF_DaughterBoard.SchDoc			

B.3 HARTING T1 Industrial Connector Daughterboard

The T1 Industrial connector from *HARTING* (figure 24) is a connector with the mating face according to the IEC 63171-6 standard [3]. It is dedicated for SPE and it is recommended for industrial applications.

Figure 24: HARTING T1 Daughterboard.

WIRE COLOURS AND PIN ASSIGNMENT

Contakt	PMA signal	Wire colour
1	BI_DA+	Blue
2	BI_DA-	White

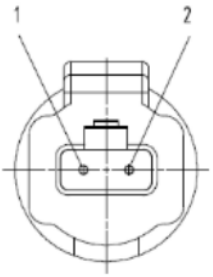


Figure 25: HARTING T1 connector drawing and pinout.

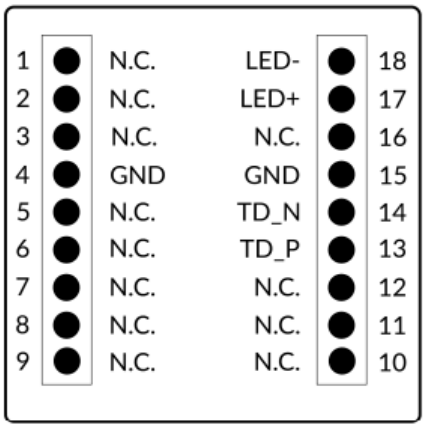
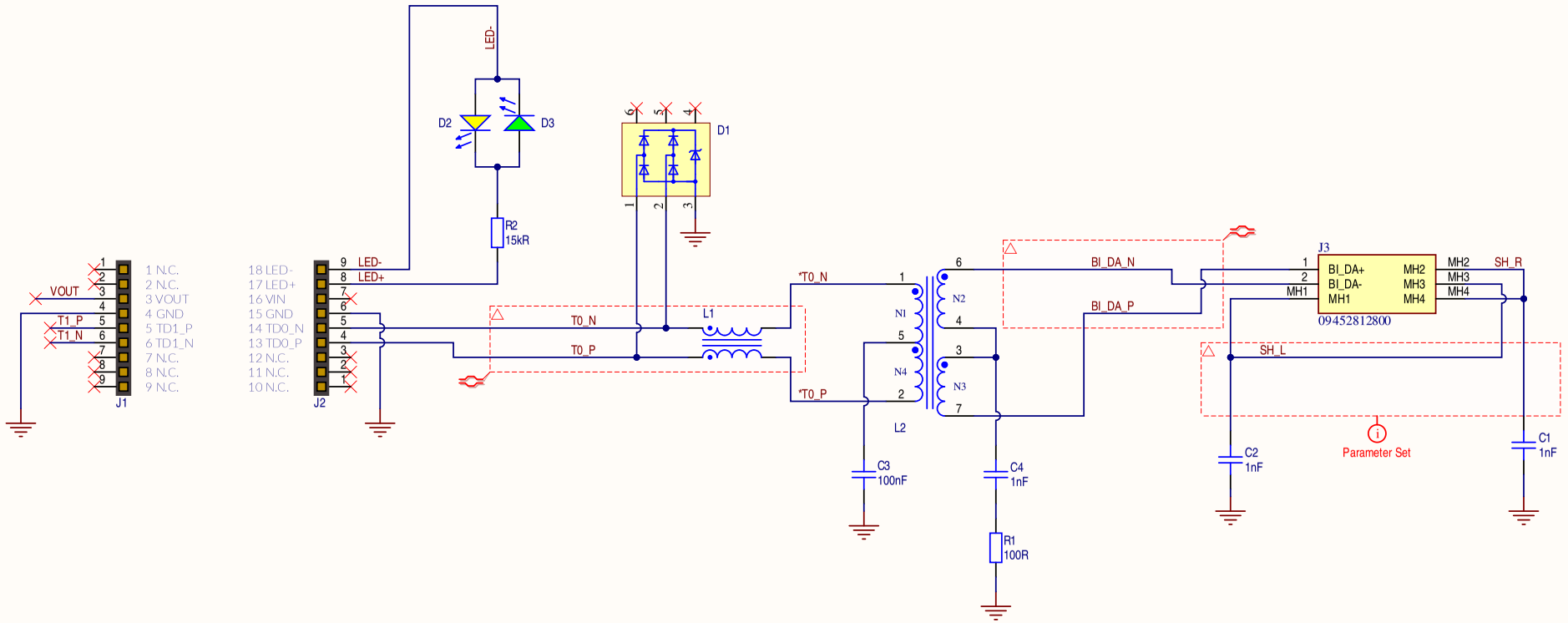


Figure 26: HARTING T1 daughterboard pinout.

- The design uses a tranformer 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.

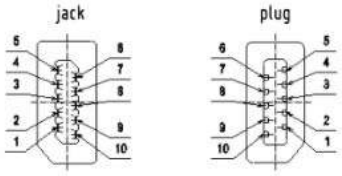


Title	T1 Industrial Connector Daughterboard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.0	Part: DaughterBoard	
Date: 23/07/2021	Time: 14:01:28	Sheet 1 of 1	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\T1_Industrial_Jack_AH_DaughterBoard\T1_Industrial_Jack_AH_DaughterBoard.SchDoc			

B.4 ix Daughterboard

The ix Daughterboard is a network daughterboard which allows using the ix connector from *HARTING* with the *periCORE Development Board*. The iX connector is an industrial Ethernet connector defined in the IEC 61076-3-124 standard [4] (see figure 27).

Figure 27: ix Daughterboard.



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 28: ix daughterboard pin assignment.

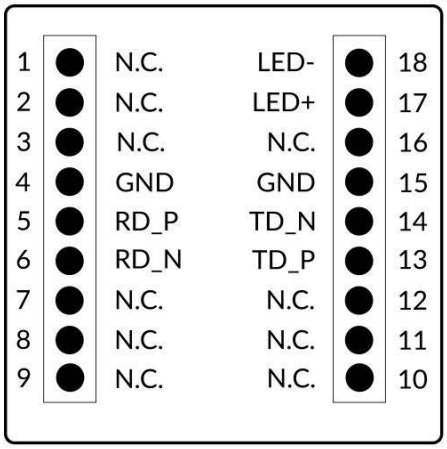
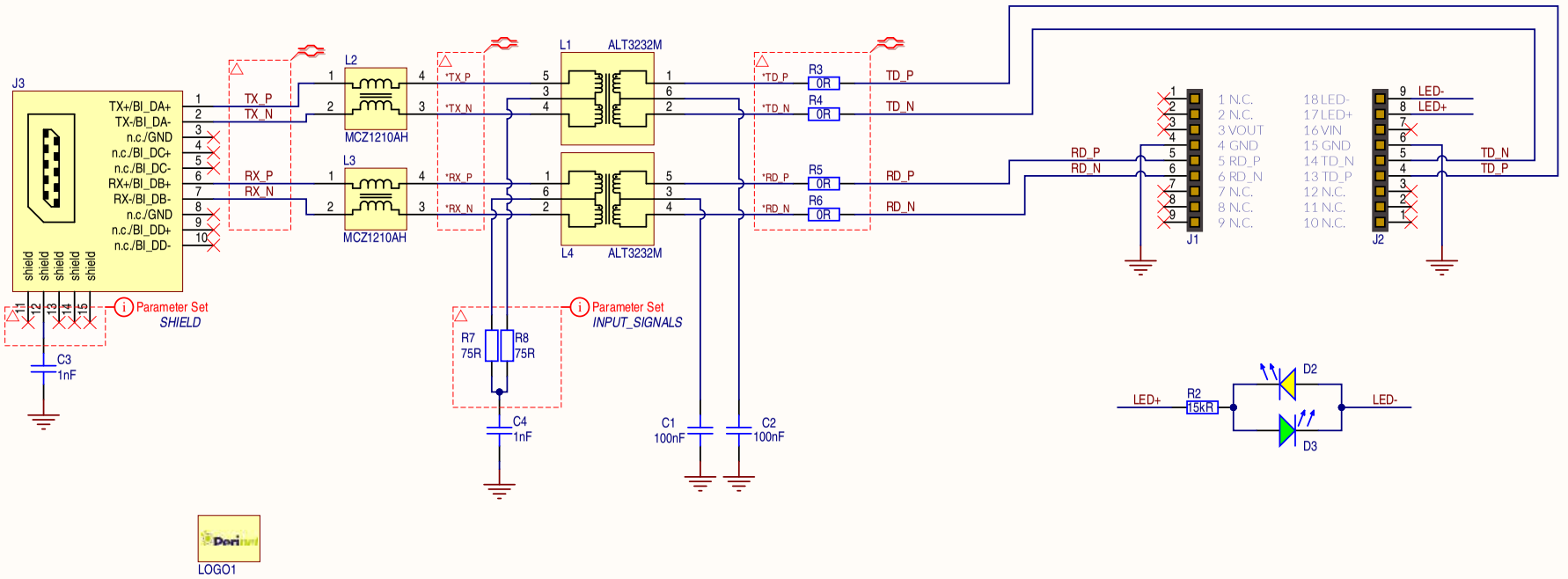


Figure 29: ix daughterboard pinout.

- 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.



Title	i.X DaughterBoard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.0	Part: DaughterBoard	
Date: 23/07/2021	Time: 16:12:47	Sheet 1 of 1	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\iX_DaughterBOard\iX_DaughterBOard.SchDoc			

B.5 RJ45 Daughterboard

The RJ45 Daughterboard is a network daughterboard which allows using the RJ45 connector with the *periCORE development board*.

Figure 30: RJ45 Daughterboard.

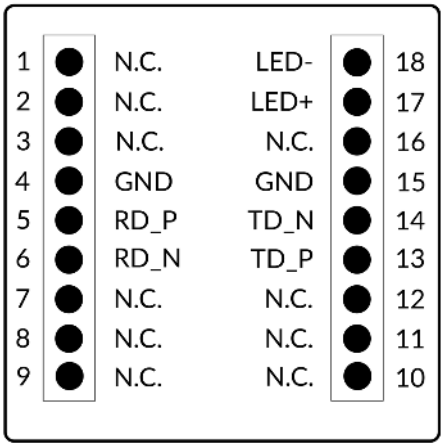
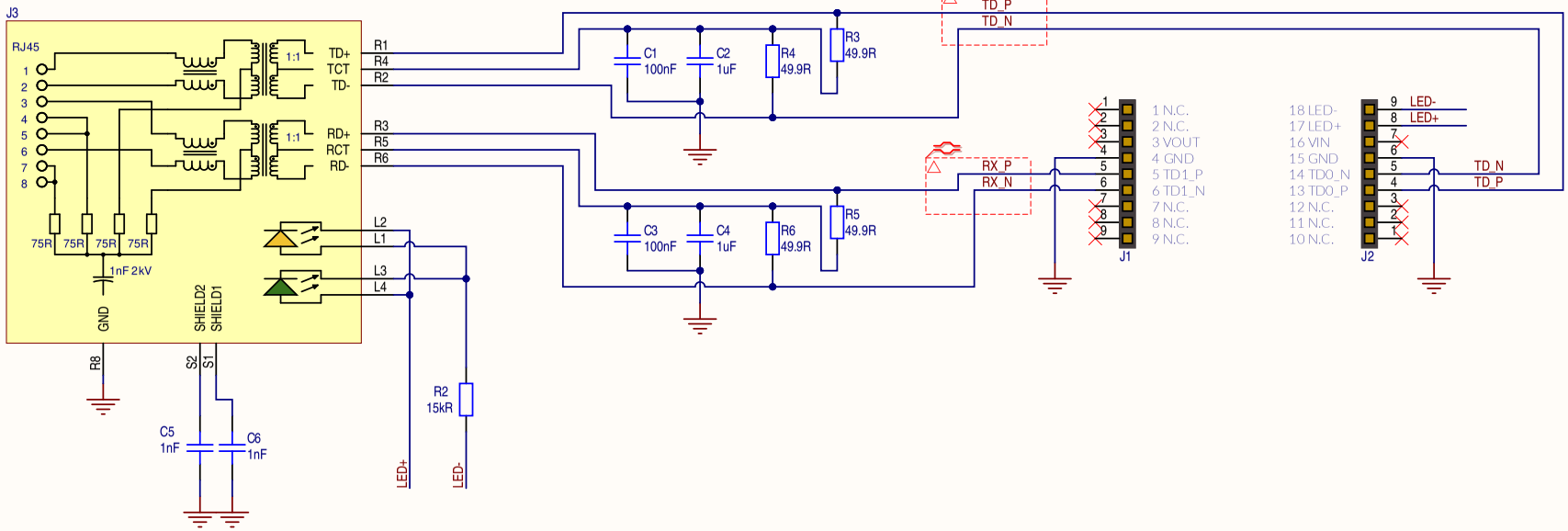


Figure 31: RJ45 daughterboard pinout.



LOGO1

Title	RJ45 Daughterboard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 23/07/2021	Time: 16:41:49	Sheet 1 of 1	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\RJ45_DaughterBoard\RJ45_DaughterBoard.SchDoc			

C Sensor/Actuator Daughterboards

C.1 PT100 Daughterboard

The PT100 Daughterboard allows the development of *periNODE-PT100* sensor applications with the *periCORE development board*. The daughterboard uses an ADS112C04, a 16-bit ADC from Texas Instruments, as an analog frontend and interface between the *periCORE* and the PT100.

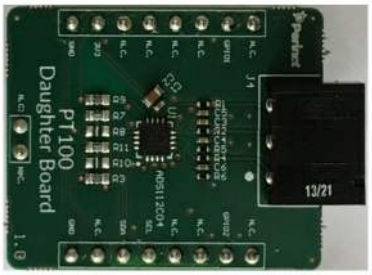
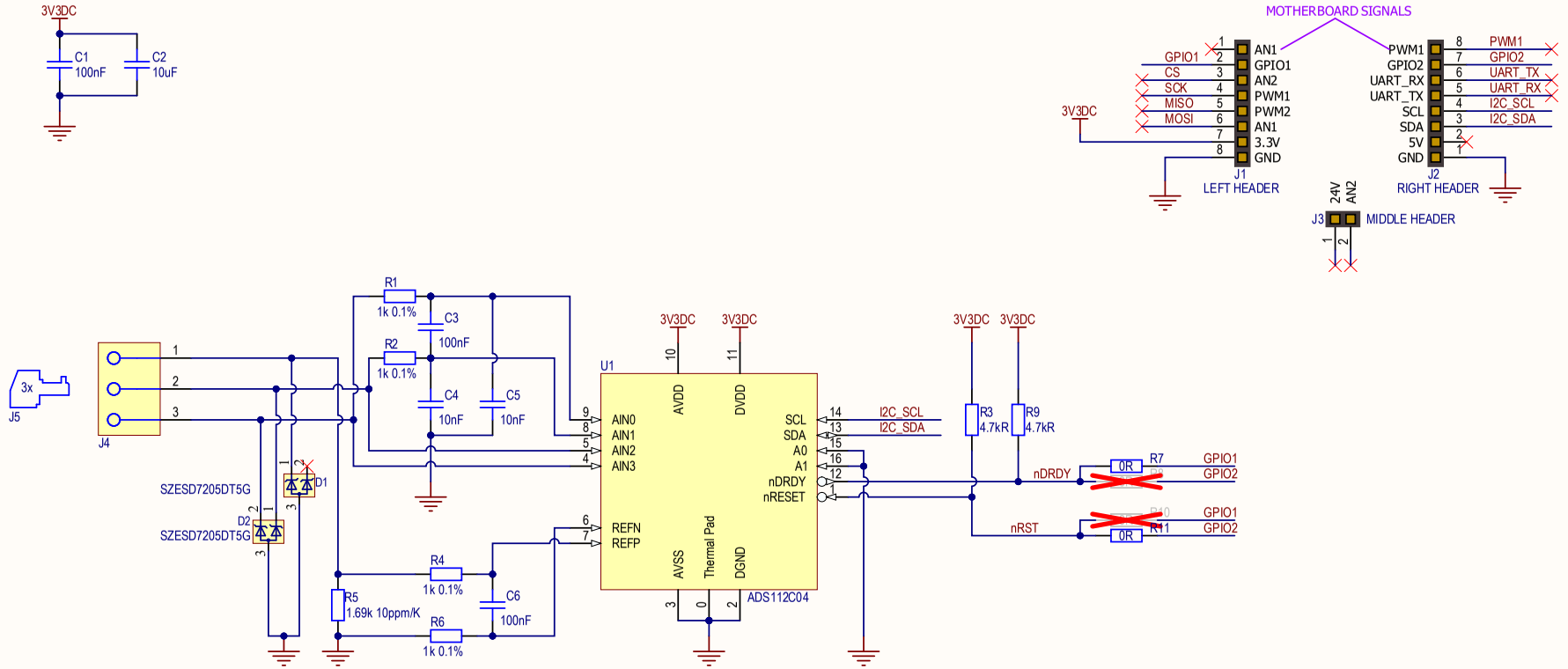


Figure 32: PT100 Daughterboard

The ciruit is designed to be used with 3-wire PT100 sensors.



LOGO1 LOGO2

Title	PT100 Daughterboard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part DaughterBoard	
Date: 26/07/2021	Time: 14:45:13	Sheet 2 of 2	
File: C:\work\Hardware_Design\periCORE_DEV_BOARD_4\PT100_DaughterBoard\PT100_DaughterBoard.SchDoc			

C.2 0-10V Daughterboard

The 0-10V Daughterboard allows connecting sensors with 0-10V analog output signal to the *periCORE* development board.

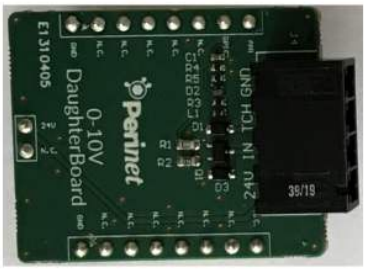
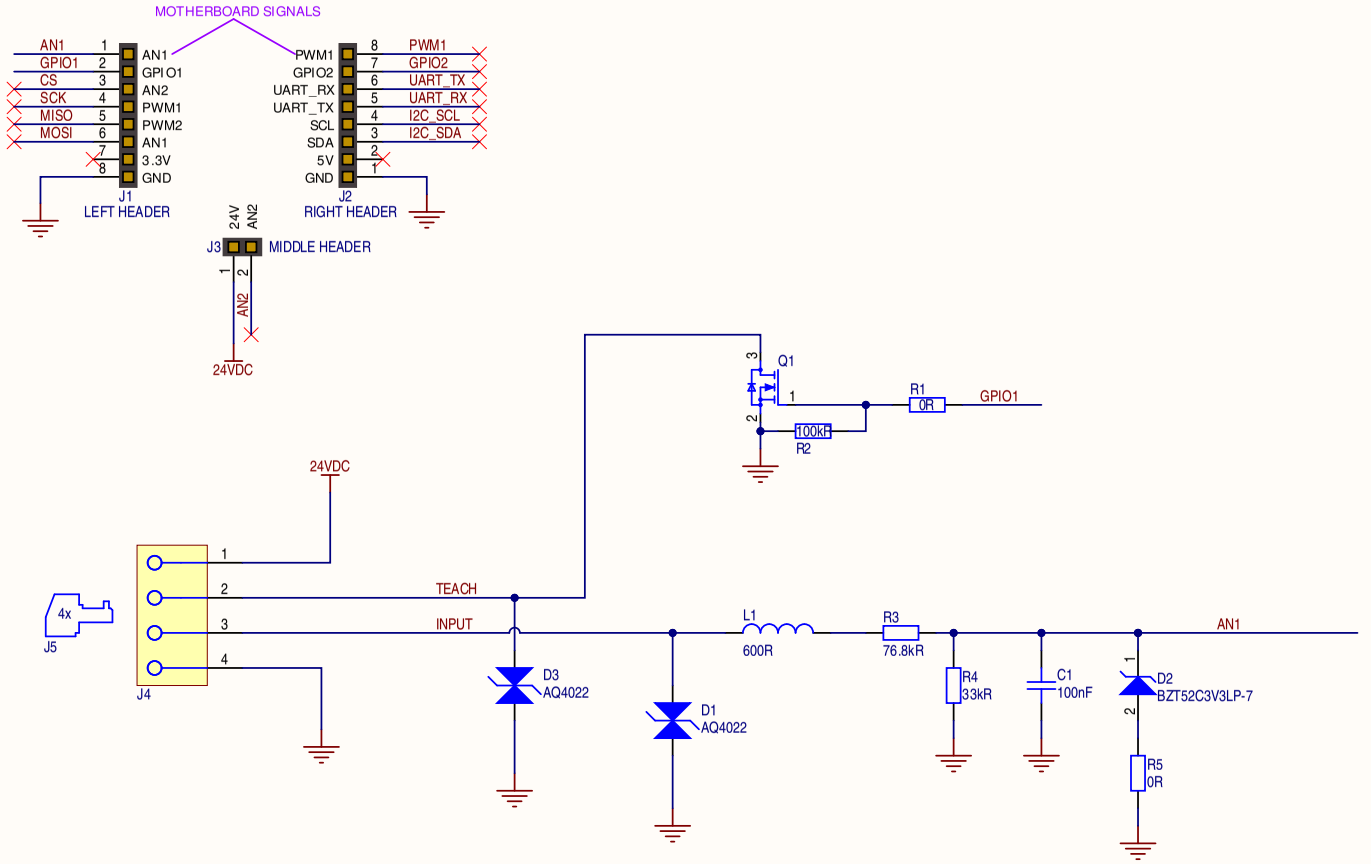


Figure 33: 0-10V Daughterboard

Signal at pin 4 of the terminal connector can be used for initiating and controling the teaching process of the sensor. It is an open-drain output driven by an N-channel MOSFET. The gate of the MOSFET is connected with the signal `GPIO1` of the *periCORE*.

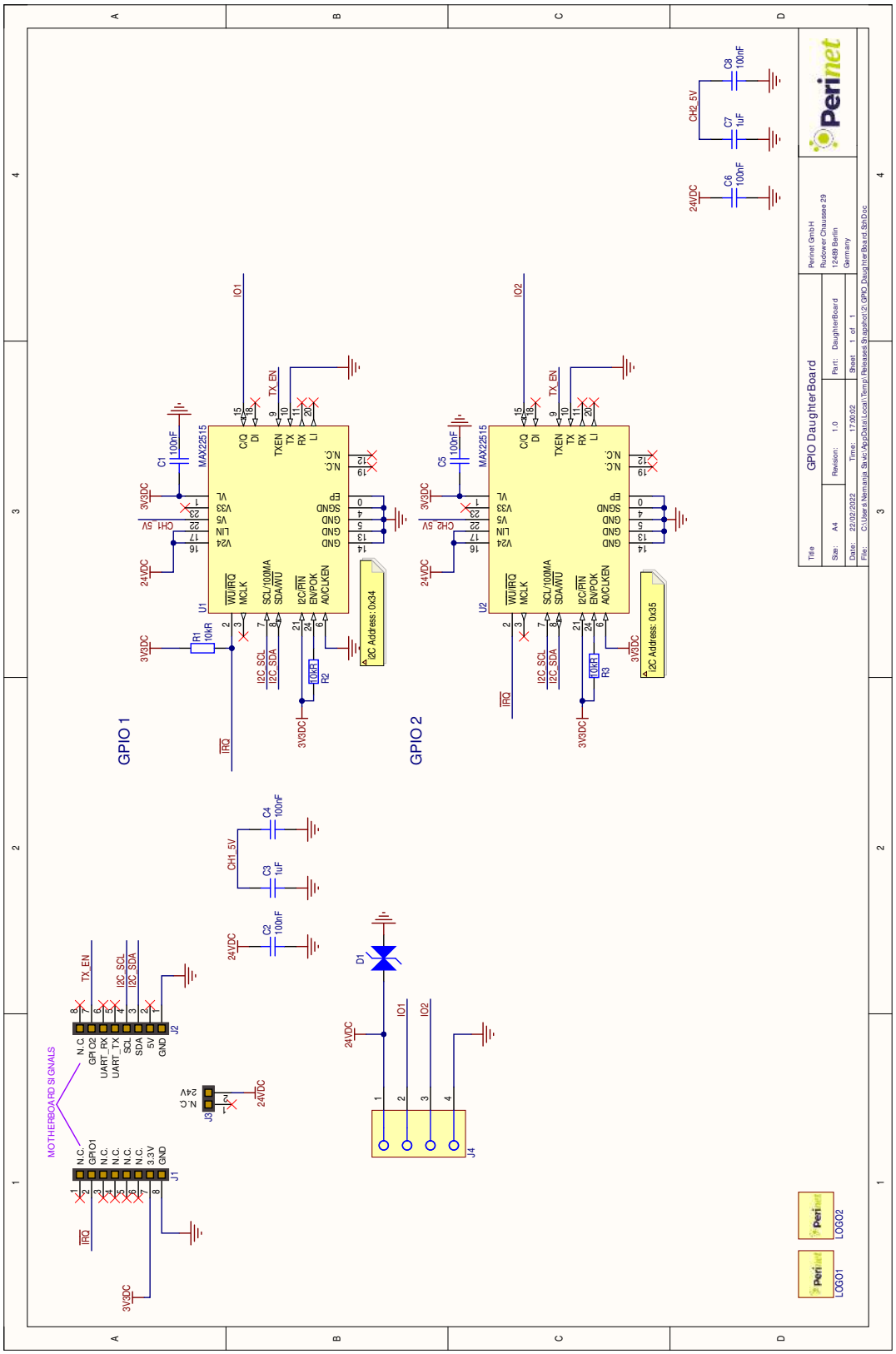


LOGO1 LOGO2

Title	0-10V Daughterboard		Perinet GmbH Rudower Chaussee 29 12489 Berlin Germany
Size: A4	Revision: 1.1	Part: DaughterBoard	
Date: 26/07/2021	Time: 21:30:52	Sheet 2 of 2	
File: C:\work\Hardware_Design\010V_DaughterBoard\010V_DaughterBoard.SchDoc			

C.3 GPIO Daughterboard

As the name suggests this daughterboard is intended to provide peripheral interfaces for *periNODE-GPIO* similar applications.



D List of Figures

E List of Listings

F 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. 10, 11, 45, 48

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". 10, 11, 54

ADC Analog Digital Converter. 58

API Application Programming Interface. 3, 14, 18–20, 25, 31

Base64 Base64 allows representation of binary data in text format.. 25, 30, 65

css Cascading Style Sheets. 25, 30, 65

DC Direct current. 9

DNS-SD DNS Service Discovery [1] is a way of using standard DNS programming interfaces, servers and packet formats to browse the network for services. 6, 20

DOM Document Object Model. 26–28, 65

GND ground, zero potential mass.. 8

html HyperText Markup Language. 25–27, 30, 65

HTTP Hypertext Transfer Protocol is an application-layer protocol for transmitting hypermedia documents, such as HTML. 14, 18–25, 30, 32, 63, 65

IIoT Industrial Internet of Things. 5

IPv4 Internet Protocol Version 4, a communication protocol. 32

IPv6 Internet Protocol Version 6 [10], a communication protocol. 19, 20, 32

jpeg An image file format. 30

js JavaScript. 25, 29, 30

JSON JavaScript Object Notation is standard text-based format for representing structured data based on JavaScript object syntax. 15, 21, 22, 24

JTAG Joint Test Action Group is a standard that defines tools to debug embedded systems. 8, 12

LED Light-Emitting Diode is a semiconductor that emits light when current flows through it. 9, 11, 48, 51, 54

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. 20

Makefile Definition of set of tasks to be executed by GNU make tool. 25, 30

MDI Media Dependent Interface, a Fast-Ethernet chipset component. 9

mDNS multicast Domain Name Service [2], a protocol that implements a local distributed name resolving mechanism. 6, 8, 15, 16, 20, 32, 63, 65

MOSFET Metal oxide semiconductor field-effect transistor. 60

MQTT Message Queuing Telemetry Transport is a lightweight, publish-subscribe based network protocol that transports messages between devices. 14, 18–20, 23, 24, 63

mTLS Mutual TLS extends the TLS protocol by requiring clients to pass certificates, allowing to provide authorization mechanisms of Application services. 20, 22, 25

OS Operating System. 17, 32

PC Personal Computer. 13

PCB Printed Circuit Board. 9

PHY Physical layer. Lowest layer of the OSI model. 11

png Portable Network Graphics - an image file format. 25, 30

RD Receive Data port. 11

REST REpresentational State Transfer, a web API style. 14, 19, 20, 25, 31

SDK Software Development Kit is a colletion of development tools. 5

SPE Single Pair Ethernet. 5, 14, 50

SPI The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for inter IC communication. 14

SSH Secure Shell Protocol. 16, 63

TCP Transmission Control Protocol. 14–17, 32

TD Transmit Data port. 11

TLS Transport Layer Security Protocol. Used by Application Protocols like MQTT or HTTP to allow secure data transfer. 20, 25

UART A Universal Asynchronous Receiver-Transmitter is a computer hardware device for asynchronous serial communication. 3, 8, 12, 14, 16, 17

UI User Interface. 5, 16, 18, 19, 25, 26, 28, 30, 31, 33, 63

URI Uniform Resource Identifier. 20–22, 29

USB Universal Serial Bus is a standard of the connection of peripherals to personal computers. 12, 14, 15

G References

[1] S. Cheshire and M. Krochmal. *DNS-Based Service Discovery*. RFC 6763. http://www.rfc-editor.org/rfc/rfc6763.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6763.txt.

[2] S. Cheshire and M. Krochmal. *Multicast DNS*. RFC 6762. http://www.rfc-editor.org/rfc/rfc6762.txt. RFC Editor, Feb. 2013. URL: http://www.rfc-editor.org/rfc/rfc6762.txt.

[3] "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* (2020). URL: https://www.beuth.de/de/norm-entwurf/din-en-iec-63171-6/321547648.

[4] "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.

[5] Perinet GmbH. periCORE Datasheet. PRN.100.375. https://docs.perinet.io/.

[6] Perinet GmbH. periCORE Development Kit Setup Application Note. PRN.100.376. https://docs.perinet.io/.

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

[8] Perinet GmbH. periMICA User Guide. PRN.100.392. https://docs.perinet.io/.

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

[10] R. Hinden and S. Deering. *IP Version 6 Addressing Architecture*. RFC 4291. http://www.rfc-editor.org/rfc/rfc4291.txt. RFC Editor, Feb. 2006. URL: http://www.rfc-editor.org/rfc/rfc4291.txt.

Revision History

Revision	Date	Author(s)	Description
1	January 9, 2022	Christian Koehler	Initial release: software development
2	January 24, 2022	Dominik Pernthaler, Christian Koehler, Adrian Schwarzer	Added hardware architecture. Updated software development section
3	March 2, 2022	Dilmari Heuer	Add Overview section
4	May 3, 2022	Christian Koehler	Add Troubleshooting and System Architecture sections. Update all other sections with latest content.