PRP-1 stack

Authors**:** S. Meier

Creation date: 08 February 2007

Last modification: 24 May 2012

Version: 1.0.7

State: released

Contact address

Institute of Embedded Systems (InES)

Zurich University of Applied Sciences

Technikumstr. 22

Postfach

CH-8401 Winterthur

Tel: +41 58 934 75 25

Fax: +41 58 935 75 25

Mail: [support.ines@zhaw.ch](mailto:support.ines@zhaw.ch)

http: ines.zhaw.ch/

Document

Distribution: ZHAW – InES

History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author/Revisor | Description |
| 1.0.1 | 08. Feb. 2007 | Meier Sven | Created |
| 1.0.2 | 16. Mar. 2007 | Meier Sven | Changed the API etc |
| 1.0.3 | 20. Mar. 2007 | Meier Sven | Updated some classes |
| 1.0.4 | 19. Sep. 2007 | Meier Sven | SRP stuff added |
| 1.0.5 | 13. Feb. 2008 | Meier Sven | Updated Graphics etc |
| 1.0.6 | 01. Jul. 2011 | Weibel Hans |  |
| 1.0.7 | 01. May 2012 | Walch Oliver | PRP-1 Updates |

Contents

[1 About this document 4](#_Toc325628790)

[1.1 The PRP-1 Stack 4](#_Toc325628791)

[2 Design 5](#_Toc325628792)

[2.1 Block Diagram 5](#_Toc325628793)

[2.2 Mode of operation 6](#_Toc325628794)

[2.3 Layer Diagram 6](#_Toc325628795)

[2.4 Components 6](#_Toc325628796)

[3 Implementation 8](#_Toc325628797)

[3.1 Object-oriented C 8](#_Toc325628798)

[3.2 API and Interfaces 10](#_Toc325628799)

[3.2.1 PRP\_T 10](#_Toc325628800)

[3.2.2 PRP\_NetItf\_T 10](#_Toc325628801)

[3.2.3 PRP\_LogItf\_T 10](#_Toc325628802)

[3.2.4 PRP\_OsItf\_T 10](#_Toc325628803)

[3.2.5 PRP\_Environment\_T 10](#_Toc325628804)

[3.2.6 PRP\_EnvironmentConfiguration\_T 10](#_Toc325628805)

[3.2.7 PRP\_Lock\_T 10](#_Toc325628806)

[3.2.8 PRP\_Supervision\_T 10](#_Toc325628807)

[3.2.9 PRP\_DiscardAlgorithm\_T 10](#_Toc325628808)

[3.2.10 PRP\_FrameAnalyser\_T 10](#_Toc325628809)

[3.2.11 PRP\_Frames\_T 11](#_Toc325628810)

[3.2.12 PRP\_Trailer\_T 11](#_Toc325628811)

[3.2.13 PRP\_Timer\_T 11](#_Toc325628812)

[3.3 Frame Path Diagrams 12](#_Toc325628813)

[3.3.1 Normal/Supervision frame receiving 12](#_Toc325628814)

[3.3.2 Normal frame transmitting 13](#_Toc325628815)

[Integration in Linux 14](#_Toc325628816)

[3.4 Example Userspace integration in Linux 14](#_Toc325628817)

[3.5 Compile and run application 14](#_Toc325628818)

[4 Graphic index 15](#_Toc325628819)

# About this document

This is a design and implementation description of the Parallel Redundancy Protocol (PRP-1) Software.

## The PRP-1 Stack

The requirements are as followed:

The Parallel Redundancy Protocol Stack shall support the whole PRP-1 standard.

It shall allow running IEEE1588/PTP.

It shall be manageable.

It shall be easy portable to other Operating Systems.

It shall be as fast as possible.

It shall use as little resources as possible.

It shall be extendable.

It shall have a clear API.

With these requirements in mind the design described in the following chapters was chosen.

# Design

## Block Diagram

**Fehler! Verweisquelle konnte nicht gefunden werden.** illustrates the relations between the different components of a PRP device.

Each PRP device owns a PRP Environment and a PRP Node Table, which is embedded into the PRP Environment. The PRP Environment provides the protocol functionalities like Discard Algorithm, Supervision or Redundancy Control Trailer as well as global and node specific data. The interfaces (PRP, PRP\_xxxItf) decouple all the operating system dependent functions from the PRP software which is generic. These interfaces are implemented as singletons what means that every interface exists only once. The access layer PRP (API) is the communication path between user programs and the PRP software.



Figure 1 Components of the PRP-1 SW stack

|  |
| --- |
|  |

## Mode of operation

Basically the following steps are executed when an event occurs:

An event occurs (e.g. a frame was received).

The respective API function gets called (e.g. receive()).

The PRP\_Environment invokes further actions depending on the current event (e.g. pass the received frame to the PRP\_Frame\_Analyser).

## Layer Diagram

PRP-1 software mainly consists of two layers. One layer is the protocol engine – it’s generic and hasn’t got to be changed at all in order to run on a specific operating system. It’s implemented according to the PRP-1 standard. The other layer is the OS abstraction layer with the interfaces – this is where all the environment dependent code is located at. The interfaces have to be adapted to the OS so the protocol engine has access to the used resources which the OS -environment provides. The layers are shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**.



Figure 2 Layers and interaction

|  |
| --- |
|  |

## Components

#### PRP\_Environment\_T

The PRP\_Environment\_T is kind of a multiplexer. It forwards API calls coming from the PRP\_API to the respective class.

#### PRP\_Frames\_T

This is the actual core of the design. It adds or removes the RCT and creates duplicates in the transmission path.

#### PRP\_FrameAnalyser\_T

The PRP\_FrameAnalyser\_T checks the frame type, whether it is a normal, a PTP or a supervision frame, and forwards it to the respective handler.

#### PRP\_DiscardAlgorithm\_T

This module implements the discard algorithm, the actual heart of the protocol.

#### PRP\_Supervision\_T

The PRP\_Supervision\_T block sends and receives supervision frames.

#### PRP\_Timer\_T

This is the Timer needed for sending supervision frames and for duplicate detection aging.

#### PRP\_Lock\_T

This block is used to guarantee atomic access on shared resources.

#### PRP\_Trailer\_T

This module provides functions to get, add and remove the RCT.

#### PRP\_EnvironmentConfiguration\_T

The PRP\_EnvironmentConfiguration\_T provides the general information about the protocol engine defined in the PRP-1 standard.

#### PRP\_T

The PRP\_T is the API of the PRP-1 software. It encapsulates the functions of the PRP\_Environment\_T that must be called from outside. It guarantees atomic access to the protocol engine.

#### PRP\_OsItf\_T

The whole software runs in user space to get an OS independent protocol core.

#### PRP\_NetIf\_T

This PRP\_NetItf\_T module provides functions to transmit and receive frames. It is the wrapper class around the protocol core. The integration into the network stack of the operating system is done here.

#### PRP\_LogItf\_T

The PRP\_LogItf\_T is the output interface of the software.

# Implementation

The implementation of PRP-1 is done with object-oriented C. So the advantages of object-oriented analysis / design / programming can be applied (also methodology, encapsulating, reusability etc.).

## Object-oriented C

This clause shows how a C++ class is realised in C.

A C++ class:

|  |  |
| --- | --- |
| /\*\*\* PRP\_Timer class in C++ \*\*\*/  class PRP\_Timer  {  /\*\* Attributes \*/  private:  boolean enabled\_;  Unsigned32 value\_;  Unsigned32 timeout\_;  /\*\* Methods \*/  public:  PRP\_Timer( Unsigned32 timeout );  ~PRP\_Timer();  tick();  restart();  stop();  } | Constructor  Destructor |

The same class PRP\_Timer in C is realised as the following.

|  |  |
| --- | --- |
| /\*\*\* PRP\_Timer class in C \*\*\*/  typedef struct PRP\_Timer\_T PRP\_Timer\_T  {  /\*\* Attributes \*/  boolean enabled\_;  Unsigned32 value\_;  Unsigned32 timeout\_;  };  /\*\* Methods \*/  PRP\_Timer\_Init( PRP\_Timer\_T \*const me, Unsigned32 timeout );  PRP\_Timer\_Cleanup(PRP\_Timer\_T \*const me );  PRP\_Timer\_tick(PRP\_Timer\_T \*const me );  PTP\_Timer\_restart(PTP\_Timer\_T \*const me );  PTP\_Timer\_stop(PTP\_Timer\_T \*const me ); | Constructor  Destructor |

The attributes of a class are encapsulated in a structure. To avoid the struct keyword, the class struct PRP\_Timer\_T is defined as a new type: typedef struct PRP\_Timer\_T. Because there is no this pointer in C, a pointer of type of this class is passed to every function of this class. The names of the methods (or functions) of a class shall always begin with the class name. Constructor and destructor shall be named <CLASSNAME>\_Init and <CLASSNAME>\_Cleanup respectively. Some classes own functions named <CLASSNAME>\_Create and <CLASSNAME>\_Destroy. The \_Create function creates an instance of the class dynamically and the \_Destroy function destroys the instance and frees the allocated memory. These two functions also call the \_Init and \_Cleanup functions.

## API and Interfaces

### PRP\_T

This class represents the API of the PRP-1 stack and describes the functions which can be called from outside in order to impact it. PRP\_T handles inputs to the stack as function calls. Every time an API function is called, it will pass a lock saved section, so every API call is atomic. This class is implemented as a singleton therefore there is no me\* pointer.

In order to ensure an error free operation of the PRP-1 Stack, inputs have to be passed to the stack by (only) using the API functions. PRP\_T is thread save and has not to be adapted for a specific OS.

### PRP\_NetItf\_T

The Network Interface integrates the protocol engine into the network stack and is therefore the wrapper class around the OS independent protocol engine. The network interface must receive frames from the two Adapters and must be able to send to the two Adapters. All traffic to the two interfaces must pass this Interface. On receiving the OS depending receive function will call the API PTP\_T\_receive() function. In the sending path it will forward the frames coming from the upper layers to the API PTP\_T\_transmit() function. The protocol engine will call the PRP\_NetItf\_T\_transmit() function. This Interface also provides functions to change properties of the two adapters. PRP\_NetItf\_T has to be adapted for a specific OS.

### PRP\_LogItf\_T

This interface provides log macros for outputs of the protocol engine. There are several different log levels. PRP\_LogItf\_T has not to be adapted for a specific OS (but maybe for a not GNU compliant C compiler because list of arguments is not supported by all compilers).

### PRP\_OsItf\_T

The Operating System Interface abstracts functions like printf, malloc, free, memcpy etc.

PRP\_OsItf\_T has to be adapted for a specific OS.

### PRP\_Environment\_T

The Environment is the central part of the protocol engine. It distributes function calls coming from the API to the respective classes. It also runs the timers.

### PRP\_EnvironmentConfiguration\_T

The environment configuration interface contains general information about the protocol engine and its environment.

### PRP\_Lock\_T

This class provides the functions to create locked sections.

### PRP\_Supervision\_T

The Supervision class is one of the core classes it processes received Supervision frames as well as it send periodically Supervision by itself.

### PRP\_DiscardAlgorithm\_T

This class is another core class. It implements the discard algorithm specified in the standard. The decision whether to keep a frame or not is done here.

### PRP\_FrameAnalyser\_T

The Frame Analyzer checks the frame type and forwards the frame to the respective frame handler.

### PRP\_Frames\_T

The Frames class is the core class of the whole protocol engine. It adds/removes the RCT, runs the discard algorithm, etc.

### PRP\_Trailer\_T

The Trailer class provides functions to add/remove/get the RCT to/from the frame.

### PRP\_Timer\_T

The Timer class is to run scheduled task like Supervision frame transmission or handling the aging for the duplicate detection.

## Frame Path Diagrams

### Normal/Supervision frame receiving



Figure 3 Reception of a frame

### Normal frame transmitting



Figure 4 Transmission of a frame

### Integration in Linux

## Example Userspace integration in Linux

The PRP protocol engine receives the frames from the two real Ethernet interfaces, does the Duplicate elimination and forwards it to a virtual network interface which forwards the frames back to the operating systems protocol engine. Therefore all applications are talking with the virtual network device.

|  |
| --- |
| Figure 5 Linux Userspace integration |

## Compile and run application

The make file is located in the prp\_pcap\_tap\_userspace directory. Change into this directory and build the PRP-1 SW-stack by calling make. If the application gets compiled successfully, the application prp\_pcap\_tap\_userspace can be executed.

# Graphic index

Figure 1 Components of the PRP-1 SW stack 5

Figure 2 Layers and interaction 6

Figure 3 Reception of a frame 12

Figure 4 Transmission of a frame 13

Figure 7 Linux Userspace integration 14