TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DIPLOMARBEIT

A versatile networked embedded platform for KNX/EIB

ausgeführt am

Institut für Rechnergestützte Automation
Arbeitsgruppe Automatisierungssysteme
der Technischen Universität Wien

unter der Anleitung von

ao. Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Kastner
und
Univ.-Ass. Dipl.-Ing. Georg Neugschwandtner

durch

Friedrich Praus
Hyrtlgasse 18/22
1160 Wien

Wien, 18.10.2005

Abstract

The deployment of home and building automation systems allows to increase comfort, safety and security and reduce operational cost. Today such systems typically follow a hierarchical distributed approach. While control networks interconnect smart sensors and actuators, a backbone network provides the infrastructure for management tasks. Devices interconnecting these networks have a strategic role. Especially in the home domain, the integration of various control and data networks is essential for maximum benefit.

The European Installation Bus (KNX/EIB) is a popular control network designed to enhance electrical installations in buildings. It uses a proprietary twisted pair (TP) medium to interconnect devices like smart light switches and dimmers. The objective of this thesis is to design an embedded and versatile platform for ongoing development in the area of home and building automation systems with a focus on KNX/EIB TP. Besides two KNX/EIB TP interfaces, it provides RS-232, USB and Ethernet connectivity. The platform moreover has sufficient processing power and storage, enabling it to act as a "smart router" or gateway.

The thesis first presents a classification of control network devices. It then discusses the hardware and software requirements for the desired platform. A detailed presentation of its design, implementation and operation with respect to hardware and software follows.

Kurzfassung

Der Einsatz von Heim- und Gebäudeautomatisierungssystemen ermöglicht gesteigerten Komfort, erhöhte Sicherheit und niedrigere Betriebskosten. Solche Systeme folgen heute in der Regel einem hierarchischen verteilten Ansatz. Während Automationsnetzwerke intelligente Sensoren und Aktuatoren verbinden, stellt ein Backbone die notwendige Infrastruktur für Managementaufgaben zur Verfügung. Geräte, die diese Netzwerke verbinden, nehmen eine strategische Position ein. Insbesondere im Heimbereich ist die Integration verschiedener Automations- und Datennetze unabdingbar, um das Potential auszuschöpfen.

Der weit verbreitete Europäische Installationsbus (KNX/EIB) ergänzt die klassische Elektroinstallation im Gebäude durch ein Automationsnetzwerk. Er verwendet ein eigenes Twisted-Pair (TP) Medium um beispielsweise intelligente Lichtschalter und Dimmer zu verknüpfen. Das Ziel dieser Arbeit ist eine vielseitige Embedded-Plattform zu entwerfen, die für zukünftige Arbeiten im Bereich der Heim- und Gebäudeautomation mit Fokus auf KNX/EIB TP herangezogen werden kann. Neben zwei KNX/EIB TP Schnittstellen beinhaltet sie RS-232, USB und Ethernet. Die Plattform stellt darüber hinaus ausreichend Rechenleistung und Speicherkapazität zur Verfügung, um als “intelligenter Router” oder Gateway zu dienen.

Die vorliegende Diplomarbeit klassifiziert zunächst Geräte der Gebäude- und Heimautomation. Nachfolgend werden Hardware- und Software-Anforderungen für die zu entwickelnde Plattform diskutiert. Eine Präsentation des Designs, der praktischen Umsetzung und der Anwendung sowohl der Hard- als auch der Software bildet den Kernpunkt der Arbeit.

Danksagung

Ich möchte mich an dieser Stelle bei allen Menschen bedanken, die mich während meines Studiums und besonders bei der Erstellung dieser Diplomarbeit begleitet haben.

Mein besonderer Dank gilt meinen Eltern für die Förderung meiner Ausbildung und ihre ständige Unterstützung.

Weiters möchte ich meinem Betreuer Wolfgang Kastner sowie Georg Neugschwandtner für die großartige Betreung und ständige Hilfsbereitschaft danken.

Vielen Dank an Bernhard Greissing für die Hilfe bei der Platinenbestückung. Ohne ihn wäre diese nicht so schnell, einfach und perfekt gelungen. Großen Dank auch an Oliver Alt für die ständige Hilfe.

Einen herzlichen Dank auch an meine Studienkollegen Woif, Jensi, Benno und Gerd für die gegenseitige Unterstützung, abwechslungsreiche Studienzeit und das Korrekturlesen.

Contents

”Whoso shall pull this Sword forth of the stone Is rightwise king, born of all England.”

Cram, Ralph Adams, Excalibur: An Arthurian Drama (1893)

Excalibur is considered as one of the most powerful fictional swords.

1 Introduction

1.1 Home and Building Automation

Home Automation Systems (HASs) and Building Automation Systems (BASs) aim at improving interaction and communication between devices typically found in buildings. The term HAS refers to small-scale installations in the residential context whereas the term BAS denotes large functional buildings like office buildings and hospitals. Requirements, expectations and complexity are different in both domains, but for both the exchange of control data is a key issue. Small, sporadically occurring data amounts have to be transferred robustly over long distances.

The core application area is environmental control with other possibilities being integrated more and more. Devices potentially participating in HASs and BASs can be classified according to their function [37]:

- Lighting and window blinds
- Heating, Ventilation and Air conditioning (HVAC) systems, including domestic water heating
- White goods (household appliances), like a washing machine or stove
- Brown goods (audio/video or home theatre equipment, game consoles)
- Communications equipment (intercom system, telephone)
- Information processing and presentation equipment (PCs, tablet PCs, PDA)
- Security and access control
- Safety alarm system
- Elevators and sundry special domains

In an automated building or home, for example, light in a room can be turned off when nobody is present or be automatically turned on if sensors detect a human being. Functions of a HVAC system can not only be affected by the current temperature, but also by other associated factors: A HVAC system can be turned off in a particular room when a window is opened and turned on again when the window is shut.

Although this classification can be applied to HAS as well as to BAS, different priorities due to different importance for integration exist. For BAS the key driving factor is economic utility. Improved control and regulation of energy consuming devices allow to reduce ongoing costs. In a small-scale HAS increasing comfort, safety and security is decisive and saving of energy is a nice benefit. Moreover, information and infotainment systems, like the "smart fridge" [64] are becoming more important.

For both domains central and remote access to the controlled functions is of concern. A central monitoring and control centre in a BAS can further reduce costs, by providing services allowing detection, localisation and correction of faulty conditions at an early stage with minimal effort. In a HAS the "peace of mind" sense is the driving factor. The user is, for instance, able to find out the condition of windows (i.e. open/closed) or the alarm equipment without actually being on site. Refer to [38] for a detailed classification, description and discussion of the above mentioned groups.

To provide the above mentioned services, the underlying system has to fulfil certain features. The traditional approach assumes a *three-level functional hierarchy*, split into management level, automation level and field level.

The field level forms the lowest part in the hierarchy. This level interacts directly with the physical environment by either collecting data or affecting the environment. Typical devices are sensors and actuators. The field level prepares and preprocesses data which then can be transmitted for further processing to other nodes.

The automatic control takes place at the automation level: It operates on data prepared by the field level and provides automatic control, including all kinds of autonomously executed sequences. Data is collected from the field level, processed and transmitted to other nodes.

At management level, information from throughout the entire system is accessible providing a global view of the whole system. Supervisory control and management of the automation functions is performed. The management level may also include logging and backup servers to collect and archive statistical data. Additionally, interconnections to other networks are also possible at this level.

Horizontal communication denotes communication among devices at the same level. If the data is transmitted between different levels, this is called *vertical communication* [65]. Normally automation systems are designed distributed. Communication is quite often performed in a peer-to-peer manner. The lack of a central controlling instance avoids introducing a single point of failure and bottlenecks. It can be observed that the amount of data to be transferred increases at higher

levels. At field level the exchange of control data is of major concern. Small packets have to be transferred and therefore the needed bandwidth (few KBits/s) is significantly smaller than the bandwidth (several MBits/s) at higher levels. Each level of the presented three-level hierarchy fulfils a certain function. On the other hand each device of an automation system implements a particular function. The trend goes towards "intelligent" devices implementing functionality from more than one of the presented levels. A smart sensor, for example, implements more and more functions of the automation level, too. However, it is still not able to fulfil the requirements of management level. Therefore the three-level functional hierarchy is implemented as a flatter two-level architecture consisting of control network and backbone network.

An automation system can consist of many hundred control network devices interacting with the surrounding and communicating among each other. They form the lowest part in HAS or BAS hierarchy and perform more and more logical functions. Due to the mass of devices, component prices have to be low and treatment should be easy: Installation and configuration have to be as simple as possible. Technologies such as link power[1] should be utilised. Earlier, sensors and actuators were simple and not very powerful. Parameters of a sensor, for instance, have traditionally been preset and fixed in hardware, but due to increasing processing power and miniaturisation of electronic devices, they are made "intelligent" by equipping them with microprocessors. This allows to alter the behaviour if necessary. Manifold possibilities arise and great flexibility can be achieved, not only in the business/industrial domain, but also regarding smart homes. However, the network stack, messages and overhead still have to be kept small.

Typical representatives of Control Networks (CNs) are the LONWorks (LON) standard [6, 7, 8] described in [44] and KNX/EIB (see [40, 21]). CNs should transfer local control data robustly under relaxed timing requirements and over long distances. They interconnect actuators and sensors at room level. At the electrical level CNs should permit easy cabling with a flexible network topology. Furthermore, large cable lengths should be possible. Communication is quite often performed in a peer-to-peer manner. Limited resources are present at field level nowadays. Therefore it is not economically feasible to use IP at this low level yet.

The backbone network is responsible for interconnecting control networks. The communication system features high bandwidth. Quite often an IP network is used, frequently employing the Building Automation and Control Networking

[1] Using link power, communication and power signal are transported via the same cable.

Protocol (BACnet) standard [9, 34] (for description see [41, 12]). Devices connected to the backbone network have comparatively high computational power and form central parts of the system. Refer to [38, 23] for a detailed discussion on communication systems for building automation and control.
It has to be mentioned, that special care has to be taken concerning safety critical devices/systems, like fire alarm systems, due to their high dependability requirements. A non working or just sporadically working light can be annoying, but no harm is caused. A malfunctioning fire alarm system in a HAS or BAS can cause catastrophic consequences. For this reason, safety critical systems are often kept separate from other building control systems.

1.2 KNX/EIB overview

The European Installation Bus (KNX/EIB) is designed to enhance electrical installations in HAS and BAS. It is a typical representative of a CN and it is based on an open specification. Konnex (KNX) is responsible for centrally maintaining the specification as well as managing Intellectual Property Rights (IPR) concerning the standard and the involved companies. KNX/EIB is well-established in central Europe, i.e. Germany, Austria and Switzerland. It is found primarily in large building installations with main application areas being lighting, control of window blinds and HVAC systems. Due to rather high component prices, the pure cost-benefit calculation regarding saving of energy does not seem to be economically feasible in home area [48, 46]. Here increase of comfort, safety and security is the major reason for using KNX/EIB.

KNX/EIB features a decentralised design. It is a peer-to-peer network system: Nodes[2] communicate directly with each other using a distributed algorithm for medium access. There are hardly any central control nodes that solely fulfil regulation functions as, for instance, a Programmable Logic Controller (PLC) does. In fact, control and working logic is located at every single node. To put it differently: Every single sensor or actuator implementing the KNX standard is able to handle network communication as well as implementation of the desired logic on its own.

A key feature in KNX/EIB is group communication based on a publisher-subscriber model, allowing to address an arbitrary number of receivers by way of a single message. A sender uses a logical group address as its destination address. Receiving stations know their dedicated group (or groups) and can accordingly

[2] A node can be anything from a simple sensor/actuator to a PC-based management server.

ignore or process incoming messages. Hence, a sender does not require information about – it cannot even determine – which nodes actually are receivers of a message. This addressing scheme is present in *process data exchange*. Only for *configuration and management purposes* point-to-point messages are used.

KNX/EIB allows various media. The primary used medium is shielded or unshielded Twisted Pair (TP) cabling known as *KNX TP1*. 29 V DC power and the actual signal are carried by the cable. Data is transmitted at 9600 b/s. Medium access is controlled using Carrier Sense Multiple Access (CSMA) with bit-wise arbitration on message priority and node address. TP1 allows a free topology with cable lengths of up to 1000 m per physical segment. A hierarchical three or two layer structure is possible using routers. See Section 1.3 for a detailed classification of devices. Integration of KNX/EIB into existing installations can be achieved using *powerline* as communication system. The mains distribution cables are spread well over the entire building and can constitute a sub-network to the communication system. Unreliability of the medium, however, requires sophisticated methods at protocol level. Speed is limited to 1200 b/s respectively 2400 b/s depending on used technology.

To further extend KNX/EIB, *KNX Radio Frequency (RF)* can be used. A subband in the 868 MHz frequency band, reserved for short-range devices, is used. For a detailed description of KNX/EIB refer to [37], for an introduction to KNX/EIB interfaces refer to Section 2.

1.3 KNX/EIB device classes and market overview

The following section is going to give a brief introduction into KNX/EIB device classes. Different requirements – technical ones and practical ones –, various abstraction levels and complexity levels can be considered to divide KNX/EIB related devices into classes. The position in hierarchy and functions of the developed hardware should be explained and the requirements like memory, storage and processing power should be determined and justified.

Table 1 shows a classification consisting of four categories which are most suitable for this thesis. The following aspects are discussed:

- Devices
- Examples: available devices
- Level: classification according to the three-level functional hierarchy presented in Section 1.1

- Function: general purpose of devices
- Installation domain: location, where devices are typically found
- Processing power
- Memory: volatile memory (e.g. RAM)
- Telegram rate: Average load (e.g. number of packets a device has to be able to process) under typical operating conditions
- Response time of the device: timing requirements regarding user interaction (e.g. time between pressing a switch and a light going on)
- Connectivity / network stack: number of required network connections and size of according network stack(s)
- Storage: permanent storage (for e.g. filter tables, historical data, ...)
- Application size: complexity and size of corresponding software (e.g. application logic in sensors, visualisation tool, ...)
- User interface: possibilities for users to interact with the system (e.g. light switch, terminal for PC-based devices, ...)
- Costs

Furthermore, some state of the art devices and systems will be presented in this section. A further introduction to home area network technologies (Bluetooth, Firewire, ...) is given in [2].

Class	Interaction devices	Routers	Gateways	PC-based
Devices	sensors, actuators	couplers, routers	gateways	control and monitoring systems, visualisation systems, logging servers

Examples	4-way binary switch, ...	line-/area coupler, EIBnet/IP router, ...	ISDN, Bluetooth, HTTP, WAP, Firewire, ...	Gira Homeserver, ...
Level	field	field	automation	management
Function	environmental control	network bridging	network interconnection	central control/maintenance
Installation domain	field/cabinet	cabinet	cabinet/office	office
Processing power	low	low	middle	high
Memory	low	low	moderate	high
Telegram rate	low	moderate	low/moderate	high
Response time	critical	critical	critical/moderate	low
Connectivity / network stack	single and small	single and small / 2 and moderate	2 or more and big	2 or more and big
Storage	low	low/moderate	moderate	high
Application size	moderate	low	moderate	high
User interface	simple	none	none/simple	complex
Costs	low	low	moderate	high

Table 1: KNX/EIB device classification

1.3.1 Interaction devices

Devices belonging to this class provide the function of an automation system. In KNX/EIB they contain part of the system's logic and functionality. The domain of interaction devices is stretched widely. What possibilities do current systems, especially KNX/EIB offer? HVAC systems have been the first devices to be auto-

mated due to resulting lower building energy consumption. Main application areas are in industrial complexes and business buildings. However, flexible HVAC systems are finding their ways into the home automation area. Single room controlled heating, cooling and ventilation solutions are possible and affordable. Artificial lighting, shading and shutter systems, centrally or de-centrally controlled, can be realised. Supervision of doors, windows or even Residual Current Device (RCD) devices can be achieved with simple binary sensors and analysed centrally. What is more, integration of safety systems such as fire detection sensors, alarm equipments or handicapped emergency call systems is possible. Visit, for example [49] or [48] for off-the-shelf available devices.

1.3.2 Routers

The catch-all term router is used in this thesis for devices interconnecting networks which share the "same protocol": Routers transfer messages, interpreting and altering them up to layer three of the ISO/OSI model.

The most primitive routers in KNX/EIB are *couplers*. They make the hierarchical structure of the system possible by connecting electrically independent parts of an installation for data transfer. Filtering by use of special tables in couplers reduces telegram traffic throughout the installation. In KNX/EIB different types of couplers are distinguished according to their position in hierarchy and used medium. Couplers for the same medium (e.g. KNX TP1 - KNX TP1) as well as couplers for different media (e.g. KNX TP1 - KNX powerline) exist. Their function, however, is mostly identical and therefore only KNX TP1 - to - TP1 couplers are discussed.

All types of couplers contain microcontrollers and RAM for filtering tables and message buffers. Filtering tables can be programmed via software[3]. *Line repeaters* have been used in the original EIB specification (also called TP64) to extend the maximum line length and maximum number of 64 devices per electrical segment. With the current KNX TP1 specification, up to 255 devices can be connected to a line and repeaters are only used to extend the line length to up to 4000m. Different lines are connected by *line couplers*. Up to 15 lines can be connected to form a main line. Typically such main lines form a control network on every floor of a building. By the help of *backbone couplers* up to 15 lines can be connected to the backbone line.

[3]Filter tables can be programmed via EIB Tool Software (ETS).

It has to be mentioned, that these standard couplers do not overcome the limitations of CNs, which are small bandwidth and short range. To create a high performance backbone more advanced and powerful networks have to be used. Due to its widespread deployment, IP is commonly used. The first approach was to tunnel standard KNX/EIB frames in point-to-point IP frames and provide remote access through the legacy *EIBlib/IP* ("iETS") (see Section 2.3.1) protocol. For this reason special tunnelling routers located at backbone level have been used. iETS has been replaced by the EIBnet/IP standard (see Section 2.3.2), which is currently awaiting voting for inclusion into the KNX standard. It addresses point-to-point tunnelling as well as routing functions and allows local control networks to be connected by a high performance backbone network. See Section 2 for protocol descriptions.

Of course, enhanced performance and flexibility do not come for free. First of all, neither KNX/EIB, EIBlib/IP nor EIBnet/IP do address security issues in order to keep the protocol simple. So the network has to be separated from the real IP world or be secured by, for instance, a Virtual Private Network (VPN) connection (refer to [32] for a detailed discussion). What is more, timing problems arise when simply extending the network size. Troubles with Medium Access Control (MAC) as well as other timing constraints circumvent arbitrary extensions. Of course, all participants in the network must handle the CN protocol.

1.3.3 Gateways

Gateways handle interconnection between different types – meaning that network protocols differ – of networks. They convert information at application level – layer 7 of the ISO/OSI model – and hence data mappings between the different network entities have to be maintained, which obviously is not possible for all types of messages. Generally it can be stated that this mapping is limited to process data exchange (e.g. group communication in KNX/EIB). Gateways allow integration of many different devices. Connection to brown goods can be realised with, for example, a Firewire-Echonet home automation network gateway. See [63] for details. Integration of white goods like washing machines or fridges can be achieved with, for instance, technologies like serve@home or Miele@home. Quite often, such gateways are based on a web browser with possible Java or Flash plugins. In HAS the concept *residential gateway* is quite common. A single central gateway forms the basis for a tight integration of all sort of consumer de-

vices. It provides access to the outside world by, for instance, ISDN or Ethernet and solves problems like where devices are located, when devices are connected or which capabilities devices feature. To put it differently: Residential gateways provide automatic service detection, support for multiple physical network technologies and remote management and can therefore also be seen as members of the PC-based device class. A separator between the two closely related classes can be drawn regarding the user interface. Gateways typically provide additional network and protocol interfaces, whereas PC-based systems provide a rich user interface. Refer to [11] or visit for example [18].

Security is of great concern for gateways because they often connect non-secure home networks to the "bad" outside world. Various possibilities exist but the reader is left to his own resources [18, 63, 11, 32]. Of course, gateways also should provide a simple user interface and should be easy to set up. Hardware requirements are rather high due to the required processing power and memory[4]. Simple KNX TP1 - KNX TP1 couplers with, for example, an 8-bit CPU and with very limited RAM are not powerful enough. For KNX/EIB a lot of gateways already exist. See Table 2 for an overview of available devices – from compact nodes to PC-class related technologies, listed for completeness. For Internet links see Appendix A.1.

Company	Product	Function	Interfaces	EIB interface
ABB	Telefon-Gateway TG/S 3.1	control via ISDN	voice, e-mail, SMS	integrated
ABB	DALI-gateway DG/S 8.1	interconnection of up to 128 DALI devices	DALI	integrated
Adyna	IC.1 DR-EIB	visualisation and control via ISDN, Ethernet and USB	Domoport, HTTP	integrated
Amann GmbH	EWMS	interconnection of BACnet devices	BACnet, Ethernet	integrated
ASTON GmbH	iPort	KNX/EIB-ISDN-Ethernet gateway	SMS, WAP, HTTP	integrated
b.a.b-technologie GmbH	eibPort	visualisation and control via ISDN and Ethernet	SMS, WAP, iETS	integrated

[4]Excalibur is a typical gateway. It features a 16 bit CPU with 24 MHz, 384 Kb ROM and 24 Kb RAM.

Daetwyler Cables+Systems	Eiblet one	visualisation and control via Ethernet	SMS, WAP, HTTP, e-mail	BCU 2, FM (Radio module)
Disch GmbH	DISCH Gateway IP	visualisation via Ethernet	iETS, SNMP, WAP, HTTP, e-mail	integrated
ELKA Elektronik GmbH	EIB-Gateway RS232/485, DMX	interconnection of devices (e.g. PLC) via RS232/485 or of the DMX512 bus	RS232/485, DMX	integrated
Hager	Th006, Th007	visualisation and control via ISDN, Ethernet and USB	Domoport, WAP, HTTP	integrated
Albrecht Jung GmbH & Co. KG	KNX/EIB Bluetooth-Gateway	control via Bluetooth	Bluetooth	integrated
Schlaps&Partner	CCEIBSPS	PLC for EIB with ISDN and Ethernet connection	HTTP, iETS	integrated
TU-Wien	KNXcalibur	flexible, embedded KNX TP1 prototype with Ethernet, USB, RS-232 and SD/MMC card	EIBnet/IP, BASys, HTTP	2x integrated

Table 2: KNX/EIB gateways (based on [49] and [48])

1.3.4 PC-based

Gateways usually just provide a connection to a bus but can also perform simple server tasks like running an integrated web server. Complex visualisation and user-interfaces, however, are limited to PC-based class devices. They are located at a central point and obtain their data using vertical communication from all over the BAS network. Often, LAN technologies are used for this purpose. The approved way of transporting a protocol over another protocol – like for example IP over Ethernet or EIBnet/IP – is very suitable and widespread. Central PC-based access to a BAS allows easier configuration and integration of devices, visualisation of control loops as well as management. Of course, connection points for remote management or access need to be secured.

For KNX/EIB many PC-based solutions exist. As mentioned in Section 1.3.3, some gateways are very powerful and hence can also be seen as PC-based devices. They offer similar facilities as the following discussed PC-based devices. In the HAS domain two representatives of visualisations are readily available on the market. Domoport [18] is a web based service, where the gateway is located at a safe place at the provider. The user can login on the website and then a connection to his/her home is established. This way Domoport certified devices located in the home can be controlled remotely. Gira homeserver (see Appendix A.1) allows querying and controlling devices locally on the KNX/EIB bus as well as remote access through a web portal provided by Gira.

Supervisory Control And Data Acquisition (SCADA) systems, sometimes also referred to as Centralised Control and Monitoring Systems (CCMSs), were introduced in the business area to allow central processing without having to handle each device separately and without actually being on site. Possibilities with such central devices are extensive. Abnormal or faulty conditions can be detected, localised and corrected at an early stage with minimum effort. Direct access for corporate management level can be granted, which simplifies data acquisition for facility management tasks such as cost allocation and accounting. What is more, historical operational data can be gained to assist in further optimisation of control loops and saving costs. For business area applications check, for example, Iphon, NETxEIB or IT-GmbH. See Appendix A.1 for Internet links to the above mentioned products.

1.4 Outlook on remaining sections

- Section 2 gives an overview of current technologies and interfaces to connect to the KNX/EIB bus.
- Section 3 describes the requirements, constraints and initial thoughts about the gateway.
- Section 4 depicts the design and outline of the hardware.
- Section 5 outlines the implemented software layers and firmware.
- Section 6 is the place to look for thoughts about further extensions, improvements and optimisations.

2 Interfaces

This section provides a survey of current technologies and interfaces to connect to the KNX/EIB bus. Depending on the application, these possibilities have various advantages as well as downsides. Selection of the appropriate interface depends on available resources (Central Processing Unit (CPU), memory, ...) of the application, desired control over the bus and abstraction level. Various protocols have to be implemented, which differ in complexity (stateful, stateless,...) and memory requirements. Especially response time requirements of the application regarding network communication have to be considered.

A typical KNX/EIB device usually consists of three parts:

- Bus Attachment Unit (BAU): Responsible for providing a connection to the KNX medium. It usually consists of a transmission unit, memory (Read Only Memory (ROM), Random Access Memory (RAM), Electrically Erasable Programmable ROM (EEPROM)) and a microcontroller.

- Physical External Interface (PEI): Standardised[5] interface providing a well-known attachment point for interconnection to the BAU. Various modes of communication exist – from simple digital I/O to synchronous/asynchronous serial protocols.

- Application module: Forms the user visible part of a KNX/EIB device. The term "application module" covers devices like push buttons, motion sensors, RS232 interfaces or USB interfaces. Depending on the used device, a different application program can be loaded to the BAU to make the device work. Various application module types are defined in KNX/EIB, which can be detected in hardware with the help of a special resistor.

Section 2.1 gives a survey of specified serial interfaces and BAUs. Bus Coupling Unit (BCU) and Bus Interface Module (BIM) (Section 2.1) based solutions can be used as a stand alone device: Layer 7 of standard ISO/OSI model can be implemented directly on the device. Simple application modules (sensors, actuators, ...) can be connected to the BAU via the PEI as digital or analog I/O. Due to limited resources on both types of BCUs/BIMs, the associated application programs have to be kept very small. If the available resources are insufficient, three possibilities exist to interconnect own applications using the BAUs: RS232

[5]The interface is standardised in hardware as well as in basic transport protocol. 21 different types are specified.

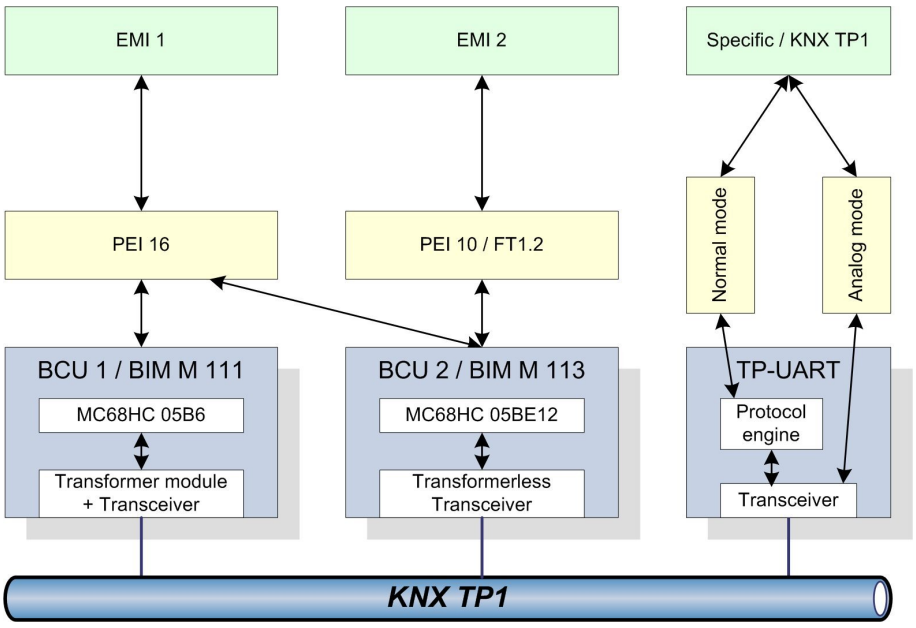


Figure 1: KNX TP1 BAUs

is discussed in Section 2.1. It is continuously used due to its simplicity regarding implementation. Section 2.2 gives an introduction to USB and the KNX on USB protocol and Section 2.3 describes standardised IP interfaces.

2.1 Serial interfacing

For this section only the asynchronous serial protocols of the PEI and the RS232 interfaces as application modules or their corresponding counterparts (on e.g. TP-UART) are relevant. Response time, protocol and hardware requirements will be discussed in a way that allows the appropriate BAU for an application like this platform to be selected. Moreover, the actual external messages being transported via the PEI are presented. All external messages together form the External Message Interface (EMI), for which currently the versions EMI 1, EMI 2 ([40] Part 3/6/3) and common External Message Interface (cEMI) ([40] AN033) exist. They differ in layer access management[6], available services and service encoding. The

[6]Layer access management allows an application to directly access a KNX communication layer (application layer, network layer, . . .).

cEMI format is intended to be used for the next generation of KNX devices and will not discussed here.

To connect the BAUs via Universal Asynchronous Receive Transmit (UART) to the external user application, two possibilities are common:

1. TTL logic level: The BAUs can be integrated into the user application and be directly connected to the hardware. TTL logic level is used.

2. RS232: The BAUs are connected via a serial cable. EIA-232, also known as RS232 forms a widespread and common standard [20]. A RS232 level converter needs to be attached to the UART and communication can take place. Operating Systems (OSs) support RS232 communication via the COM interface. Speed is limited, for KNX/EIB communication, however, it is quite sufficient.

Table 3 gives an overview about capabilities of the different BAUs. Details are discussed in Section 2.1.1, Section 2.1.2 and Section 2.1.3.

	BCU 1 / BIM111	BCU 2 / BIM 113	TP-UART
layer 1 access	✗	✗	✓
layer 2 raw access	r	r/w	r/w
layer 7 support-standalone	✓	✓	✗
UART interface	✓	✓	✓
wires	5	3	2 / 3
required response time for com. partner at serial interface	<3ms[7]	<3.3ms	<2 - 2.5ms
EMI 1 support	✓	✓	✗
EMI 2 support	✗	✓	✗
cEMI support	✗	✗	✗

Table 3: Comparison TP1 bus attachment units

[7]To achieve a constant transfer rate of 9600 bps, a response time of <1 ms is required for toggling the CTS/RTS lines.

2.1.1 BCU

The Bus Coupling Unit (BCU) is a well-defined BAU fulfilling the specified standard in ([40] Part 09/04/01). It serves as a modular mounting platform for application modules and may contain a specific, rather simple application program or just serve as a bus interface. In any case, an OS handling part of the KNX/EIB communication is present. On the one hand this simplifies bus access but on the other hand the user may not have the desired control over the bus. BCUs usually are intended to be flush mounted and inserted into wall-boxes.
Two different BCU types are defined. The *TP1 BCU 1* features a MC68HC05B6 or compatible type CPU running at 2 MHz with 176 bytes RAM, 256 bytes EEPROM and 5936 bytes ROM.

It supports the serial asynchronous *PEI type 16* protocol to transfer messages between the external user application and the BAU's communication stack. A 5-wire connection with the lines RxD, TxD, CTS, RTS and 0 V is used. The protocol consists of four phases:

1. Hardware handshake - communication request: The communication request is a request/response protocol making use of the RTS and CTS lines. The sender initiates a connection by setting the RTS line to 0 and waiting for the receiver to lower the CTS line. Then data is transmitted. This handshake takes place on each octet transfer.

2. Software handshake - transfer of length octet: This handshake takes place on the first octet exchange and is used for determining the communication direction. Both communication partners transfer the length of the data requested to send or FF_h, if no data needs to be exchanged. In case of simultaneous requests, the BAU is considered as master.

3. Data exchange: The communication initiator sends its data octets and the receiver responds in parallel with 00_h octets at speed of 9600 bps (8 data bits, no parity bit, one stop bit).

4. Pause: After transfer both communication partners have to wait for a 3 ms timeout.

Overall communication speed in PEI type 16 protocol is controlled by help of the hardware handshake. The individual octets, however, are transferred at a fixed speed of 9600 bps. Apparently no hard response time requirements are present. Only the line timeout of 3 ms has to be detected. The data exchange process

implicates that no duplex transfer of messages is possibly, meaning that a communication partner can either receive or transmit at a time, but not both.

Messages transferred via the BCU 1 PEI are in *EMI 1* format, where the actual message format depends on the used PEI type. In EMI 1 any KNX protocol layer can be switched on or off, so that the desired layer can be accessed directly. This is achieved, by writing directly to the BCU's "system status" memory location (1 byte) with a `PC_SetValue.req` or `A_Memory_Write.req`. It has to be mentioned that only some combinations of layer selections make sense and that some layers have to be turned on to make, for instance, the internal user application work. Activating the bus monitor, which passes every KNX/EIB frame to the PEI, disables the user application and object servers.
The *TP1 BCU 2* uses a MC68HC05BE12 CPU running at 2.4576 MHz. It has 384 bytes RAM, 991 bytes EEPROM and 11904 bytes ROM. It is connected to the KNX/EIB via a FZE 1066 transceiver and is fully compatible to the BCU 1. All BCU 1 services, including PEI type 16, are supported.

BCU 2 supports the *PEI type 10*, allowing usage of the standardised FT1.2 (subset of [33]) protocol or a manufacturer specific protocol, which then needs a download of the appropriate counterpart to the BAU. FT1.2 provides reliable data transmission and allows data flow in both directions. It uses a 3-wire connection with the lines RxD, TxD and 0 V. Transmission is performed with 8 data bits and 1 stop bit with even parity. The transmission rate can be selected. Communication flow is controlled with a software send/confirm service: After transmission of a message, the receiver should respond with a positive confirm frame or with a negative confirm frame. Only after reception of such a message, the sender is allowed to send further messages. No critical timings are present.

PEI type 10 uses the *EMI 2* format. It fully includes EMI 1 services and further extends them. Message destination is dependent on a static redirection table. In normal mode all messages are directed to their default destination (link layer messages to link layer, application layer messages to internal user application, ...). With the help of `PEI_Switch.req` messages this layer access can be remapped, to transfer desired messages to the PEI instead of their default destination. The internal user application is then disabled. One type of service is of special interest: The `L_Plain_Data.req` allows raw data to be transmitted to the KNX/EIB, giving full access to the bus.

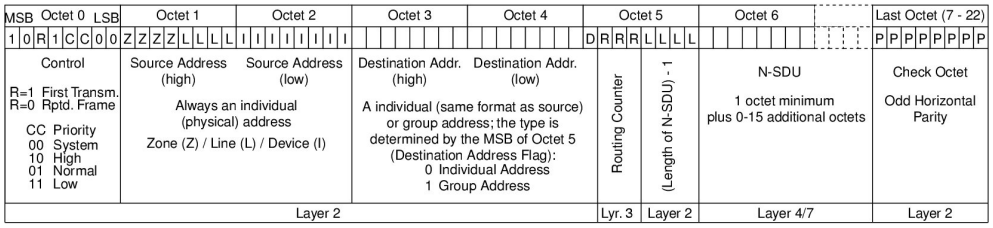


Figure 2: KNX TP1 standard data frame (from [37])

2.1.2 BIM

The name Bus Interface Module (BIM), equally to the name BCU, is used for devices complying fully with the standard in ([40] Part 09/04/02). BIMs are designed for piggyback use to be directly mounted on Printed Circuit Boards (PCBs), and hence they have no housing. The *BIM M 111* corresponds to the TP1 BCU 1 and offers the same features. It has an integrated transformer module and a controller. The *BIM M 113* corresponds to the BCU 2. It uses a transformer-less transceiver and the MC68HC05BE12 CPU. On both types of BIMs only a partially implemented PEI connector is present.

2.1.3 TP-UART

The Twisted Pair - Universal Asynchronous Receive Transmit (TP-UART) Integrated Circuit (IC) is a transceiver, which supports connecting layer 2 devices like microcontrollers, smart sensors, actuators or any other types of applications to the KNX/EIB TP1. The host controller can either be directly connected or optionally connected via optical couplers. Communication to the host controller is realised with a UART interface. It has a 64 byte telegram buffer and there are no critical timing requirements during transmission using the UART. The protocol format (i.e. message interface) is native KNX TP1 (see Figure 2), tunnelled in specific TP-UART frames. The TP-UART consists of an analog part (transceiver) and a digital part (protocol engine), allowing two different types of operation:

- Analog mode: This mode allows direct control of the EIB bus level. The TP-UART functions as a transceiver and only the analog part of the IC is working. The host has immediate control over the high and the low levels of the TP1 network, meaning that the RxD and TxD pins of the host are directly logically related to the bus level. Bit timing has to be implemented by the host.

- Normal mode: Both parts, analog and digital, are working. The host can communicate with the IC using the following constraints and protocol: The baud rate is 9600 bps or 19200 bps, depending on hardware configuration. The telegram structure is 1 start bit – 8 data bits – 1 parity bit – 1 stop bit. Each data byte transmitted to the TP-UART is prefixed with a control byte. Telegrams from EIB bus are transmitted transparently to the host, which has to detect the end of a telegram after a receive timeout of 2 - 2.5 ms. The TP-UART allows busmonitor mode and it supports the host with the Immediate Acknowledgement (IACK) service, used for confirming reception of a KNX/EIB telegram on the same electrical segment: IACKs ensure that an addressed device has received a message and is processing it.

For a detailed discussion of the device refer to specification [68].

2.2 USB interfacing

This section describes the protocol to connect a KNX USB Interface Device to KNX tools like ETS over USB. A summary of required and relevant parts of [74] is given, nevertheless the reader should be familiar with the USB specification. KNX on USB describes the tunnelling of the KNX frame formats EMI 1, EMI 2 and cEMI over USB frames. Discovery and self-description mechanisms as well as the establishment of communication links of USB are utilised.

2.2.1 Introduction to USB

The Universal Serial Bus (USB) evolved from the needs of an easy to use, expandable, low-cost, flexible, robust and fast bus to interconnect many different devices to a PC. Real-time data, such as voice, audio and video as well as “slow” human interface devices like keyboards are possible. Transfer rates of up to 480 Mb/s can be achieved.

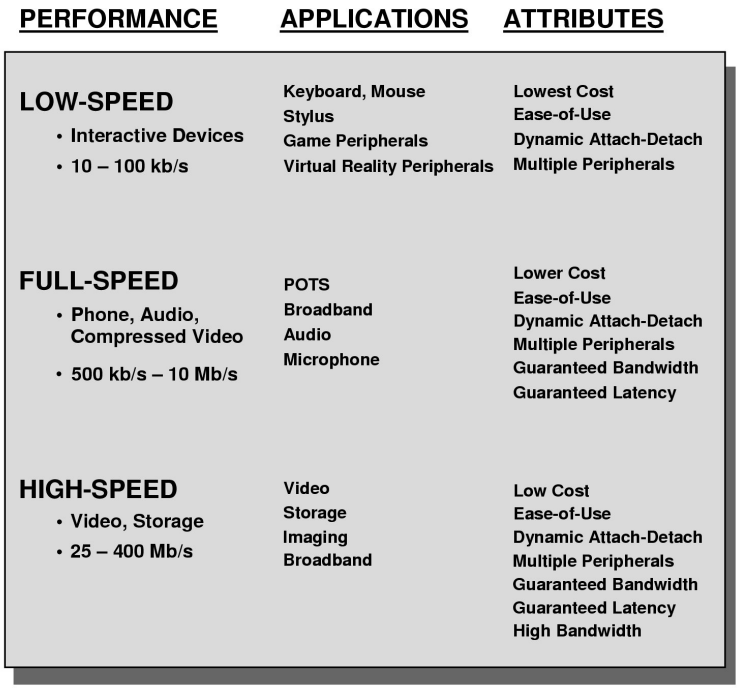


Figure 3: USB device classes (from [74])

A *USB system* consists of USB interconnect, USB devices and a USB host. Currently USB devices are broken up into:

- Hubs, providing additional USB attachment points
- Functions, providing capabilities to the system

USB hubs form the basis for the plug-and-play architecture of USB. They simplify USB connectivity from the user's point of view and expand single attachments points to multiple attachment points. A hub consists of a high speed upstream port, connected to another hub or host and several downstream ports. Architecture allows transmission at high speed at upstream port, even if full/low-speed devices are connected at downstream ports.
Functions are devices that are plugged into a port of a hub and are able to transmit or receive data over the USB. Typical functions can be seen in Figure 3 and are, for instance, keyboards or mice.

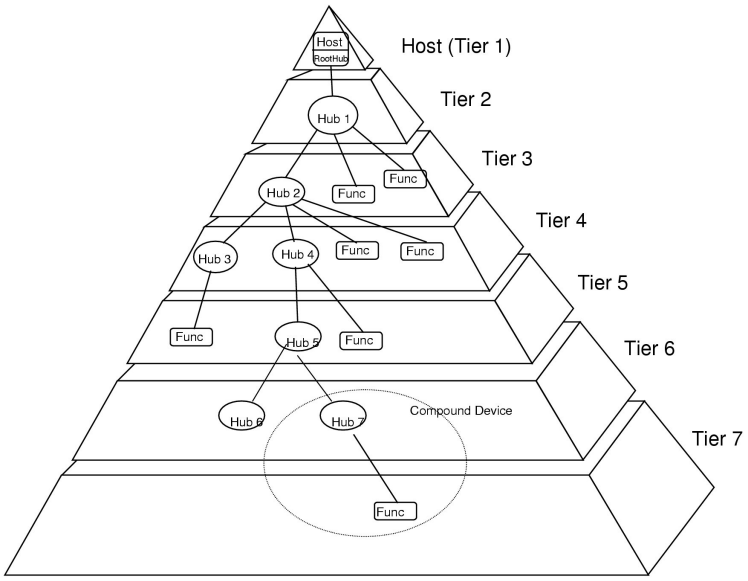


Figure 4: USB topology (from [74])

Each USB system consists of exactly one *USB host*. The USB Host is responsible for detection of attachment and removal of USB devices, management of USB standard control and data flow, collection of status and activity statistics as well as control of the electrical interface including the provision of a limited amount of power for USB devices. To provide one or more attachment points, the root hub is integrated within the host system.

The USB connects USB devices with the USB host. The *USB bus topology* is a tiered star with hubs in the centres of each star (see Figure 4). Each wire segment is a point-to-point connection between either the host and a hub or function, or between a hub connected to another hub or function.

To move data across the USB, an interaction between various layers is required. Refer to ([74] Section 5) for detailed information about physical and firmware requirements of the host and device. Here the *data flow model* required for the implementer will be presented. From a logical view, USB devices seem to be connected directly to the root port of the USB host. The different USB functions are presented to application developers and they can attach their client software to them: A so called pipe-endpoint model is used for that. The physical

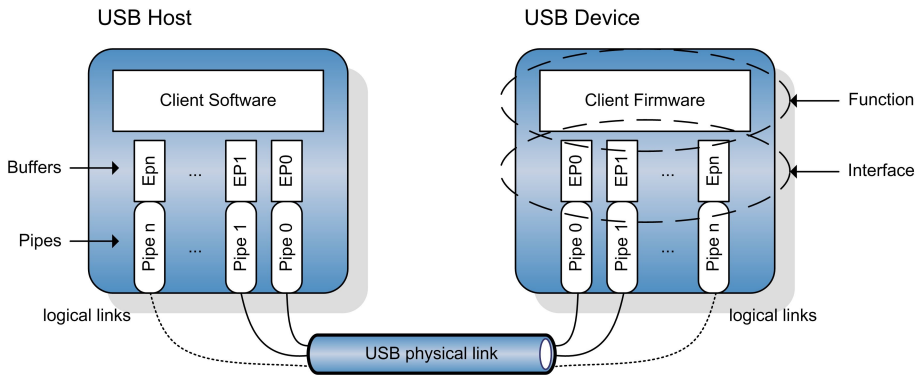


Figure 5: USB pipe-endpoint concept

link of the USB is divided into several logical links. Each logical link is represented as an endpoint featuring its own buffer/FIFO memory. A logical USB device appears to the system as a collection of endpoints. These endpoints are grouped into sets that form an interface, which in turn are the view to the function. See Figure 5 for illustration.

Endpoint 0 is associated to pipe 0 and is the Default Control Pipe, being present in every USB device. The USB System Software uses this endpoint to initialise, configure and manage the logical device. Client software uses pipe bundles and endpoint sets to manage an interface. Data is transferred from the buffer on the host to the endpoint on the device upon request of the client software. The Host Controller then packs the data, coordinates bus access time and moves data over the USB.
USB defines four *transfer types* :

- Control transfers: Bursty, non-periodic, host software-initiated request/response communication
- Isochronous transfers: Periodic, continuous communication, typically time-relevant information, guaranteed bandwidth, no retry in case of error, bounded-latency
- Interrupt transfers: Low-frequency, bounded-latency communication
- Bulk transfers: Non-periodic, large-packet bursty communication, guaranteed delivery but no guarantee on bandwidth or latency

The use of a certain transfer type and hence device class is not affected by the kind of application, but by the demands of the application concerning the underlying communication system. Devices sharing the same transport requirements can also share a single class driver because of this segmentation. Video applications, for instance, make use of isochronous transfers whereas Human Interface Devices (HIDs) have different and much simpler requirements and therefore use interrupt pipes. Combinations of multiple classes are also possible. Applications with data requirements outside this specification must provide their own drivers and class specification.

2.2.2 KNX on USB

To connect KNX/EIB to the USB ([40] AN037), two main goals have been aspired to:

- Support of only one USB Device Class to minimise compatibility problems and implementation effort
- Support of a widespread and standardised USB Device Class with host drivers available

The *Human Interface Device (HID)* class has been selected to be supported by KNX system tools. Host drivers for nearly all OS exist and hardware components are available for implementing the USB device side. Main characteristics of the USB 1.1 HID Class are interrupt transfers and maximum transfer unit of 64 octets, which are sufficient for KNX on USB. The HID Class uses two pipes – the default control pipe and an interrupt pipe – and at least 3 endpoints. The control pipe is used for standard USB requests like transferring USB control and class data, transmitting data when polled by the HID class driver and receiving data from the host. The interrupt pipe is used for receiving asynchronous data from the device as well as transmitting low latency data to the device. The interrupt out endpoint at the device is optional for HID class devices. If defined, data from the host is transmitted to this endpoint. However, if it is not defined, data is transmitted via the control endpoint. For KNX on USB the interrupt out endpoint at the device is mandatory. See Table 4 for details.

Pipe	Endpoint	Transfer Type	FIFO/Buffer length	Description
Control Pipe	EP0 In	Control	8 octets	Standard USB Requests
	EP0 Out	Control	8 octets	
Interrupt Pipe	EP1 In	Interrupt	64 octets	KNX data transfer (tunnelling)
	EP1 Out	Interrupt	64 octets	KNX local device management

Table 4: KNX HID USB class interface

In the Interrupt Pipe data is transferred in packets[8] with a maximum length of 64 octets. Requests longer than 64 octets should be split up into more than one USB HID frame. In the following, the *KNX HID frame format* will be discussed (also see Figure 6). A report frame consists of the *KNX HID Report Header* and the *KNX HID Report Body*. The header consists of the *Report ID*, which can be used by the HID Class host driver to distinguish incoming data. If a device has only one input, output and feature report structure, this prefix can be neglected. For KNX on USB, however, it is used and has a fixed value of 01_h. The *sequence number* should always start with 01_h for a KNX frame tunnelled via USB. If a KNX frame exceeds the maximum length of the KNX HID report body (61 octets), it is transmitted in multiple reports and the sequence number is incremented for every following report. Currently, the maximum number of KNX octets is 255 octets. Hence, the biggest sequence number used is five. Reports with unexpected sequence numbers should be ignored by the receiver. The *packet type* is used to determine the position of a report in a report sequence: A logical OR over the three least significant bits identifies a packet (see Figure 6). If the length of the report is, for instance, smaller than 61 octets. packet type value is 03_h (0011_b = start and end packet) because one frame is sufficient for transporting all octets. *Datalength* defines the size of the KNX HID report body.

The KNX HID report body consists of a transfer protocol header, only present in the start packet, and the USB transfer protocol body. The *protocol version* octet of the header is currently fixed to '0' with a resulting *header length* of 08_h.

[8] Also called reports (HID Spec Glossary [74]): A data structure returned by the device to the host (or vice versa). Some devices may have multiple report structures, each representing only a few items. For example, a keyboard with an integrated pointing device could report key data independently of pointing data on the same endpoint.

The *body length* is the size of the EMI frame plus one octet for message code. Since the extended frame format on TP1 allows 255 octets for KNX frames, the total USB transfer body can be greater than 255 and therefore two octets for body length are present. To identify the transported protocol, the *protocol ID* is used. Specification allows transporting of KNX, M-Bus and BatiBus frames as well as bus access server features. The latter will be discussed in the next paragraph. *EMI ID* encodes the transported EMI type. EMI 1, EMI 2 and cEMI, the future single frame format, are supported. *Manufacturer code* should be 0000_h for a KNX link layer tunnel. In case of not fully complying to the specification of protocol stated in protocol ID field, the own manufacturer's KNX member ID should be filled in. The timeout for KNX tunnelling is 1 second, meaning that the device receives a frame, transmits it to KNX medium and sends an acknowledgement within this period. If the tunnelled EMI format is not supported by the device, it is free to neglect the frame or give an error code.

For successfully connecting and recognising a KNX USB device, several parameters are of concern. Every device has a vendor ID and a product ID. These parameters are part of the USB device descriptor, which can be read by the USB host via the default control pipe. The vendor ID is a 2 byte identifier assigned by USB Implementers Forum. Companies developing USB devices can apply for an own ID or reuse existing IDs via OEM agreements. The product ID is a unique identifier assigned by the manufacturer. For use as a KNX USB device, these parameters are irrelevant. Furthermore, every device has a iManufacturer part in the USB device descriptor, which points to a human readable name of the USB device. This name, for example, shows up in the "New hardware found" dialogue of the Windows OS. Upon connecting a USB device to a Windows host, the OS reads its `vendorID` and the `productID` and searches INF files for these parameters to decide which drivers to load. Devices are grouped into various device classes by Windows. Konnex association provides the necessary values – GUID, class name and an additional descriptor[9] –, which are stored in the INF file distributed by the manufacturer with the device. Every device sticking to this interface and hence belonging to this common class fulfils the requirements to be discovered and managed by system tools like ETS.

[9]The class GUID is fixed to {01F95DC2-D064-47bc-83DD-19CE43587D2E} and is handled internally by Windows. The class name is "KNXNET" and additional descriptors are "ProviderName", "ManufacturerName", "DeviceDesc" and "DeviceClassName"

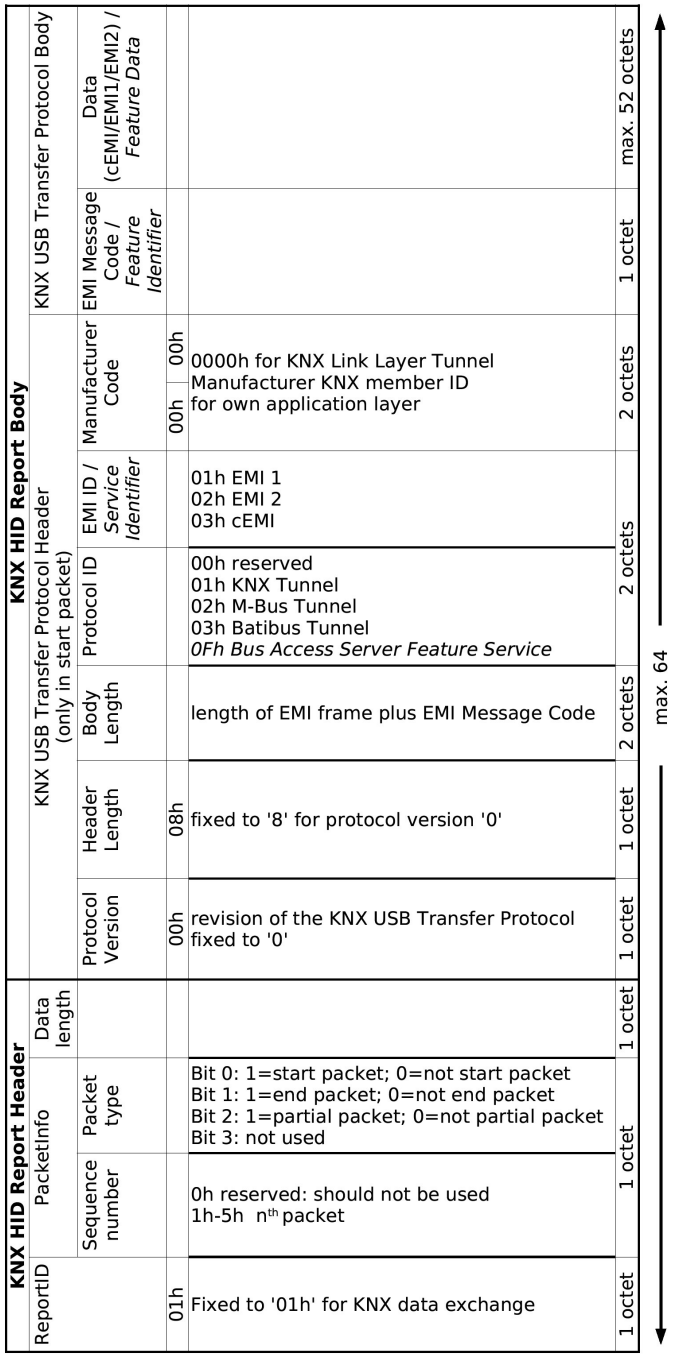


Figure 6: KNX on USB frame format

Implementations not using a cEMI server – like EMI 1 and EMI 2 on BCU 1 or BCU 2 – do not support discovery or management functions. To overcome this limit, a dedicated *bus access server feature protocol* has been proposed: Features of a device should be detected and managed separately and stateless by so called *device feature services*. cEMI capable devices should also support these messages. As seen in Figure 6, the bus access server feature uses the protocol identifier $0F_h$ of the KNX USB transfer protocol header. The selected device feature service is identified by the 1 octet field "service identifier". Possible additional data can be attached to the end of the whole message. The following feature services, listed with identifiers, exist:

- Device feature get (01_h): This feature is the only confirmed service. A device feature response should be transmitted within 1 second after reception of the get request.
- Device feature response (02_h): A variable length of data is supported. Length can be gained from the body length field decremented by one.
- Device feature set (03_h): This frame is transmitted exclusively by the bus access client to set a parameter of the bus access server.
- Device feature info (04_h): This frame is transmitted by the server and is not confirmed by the bus access client.

Currently there are five device features, listed with feature identifier, defined:

- Supported EMI type (01_h): This type is used to get the supported EMI type(s). The value is encoded in 16 bits – bit 0 determines EMI 1 support, bit 1 EMI 2 support and bit 3 cEMI support. For each supported type, the corresponding bit should be set to 1. The KNX USB access client should determine the supported EMI types upon first connect. If the desired EMI format is not supported, the client should not transmit any tunnelling frames. If the server supports more than one type, the client should select the desired format.
- Host device descriptor type 0 (02_h): With this feature, the supported local device management procedures can be obtained. This feature is most useful for BCU 1 and BCU 2 based KNX USB devices, because cEMI allows management itself.

- Bus connection status (03_h): This feature is a one-bit read only value. If connection to the bus is available, this bit should be 1, otherwise it should be 0. After change of this value, a device feature info frame should be submitted by the server.
- KNX manufacturer code (04_h): The 2 octet KNX association manufacturer code should be provided here.
- Active EMI type (05_h): Management of the current active EMI type can be performed with this feature.

For cEMI based KNX USB servers, the default management procedures of cEMI ([40] AN033) server should also be supported.

2.3 IP interfacing

2.3.1 EIBlib/IP

EIBlib/IP (iETS) [62] is the predecessor of EIBnet/IP and has actually been superseded by that protocol. It is discussed here for completeness. EIBlib/IP is an extension to ETS 2 providing access to the KNX bus via the Internet Protocol (IP) [58]. ETS 2 acts as a client whereas the EIBlib/IP device, physically connected to the KNX bus, acts as the iETS server. Communication on top of IP is based on User Datagram Protocol (UDP) [56] or Transmission Control Protocol (TCP) [59]. Three different ports are used: Read and Write channel have dedicated ports called ReadPort and WritePort (default ReadPort is 50001, default WritePort is 50002). All other communication uses the default shared port 50000. All three ports can be configured using the ETS. Payload of IP packages are frames in EMI 1 message format, contained in PEI transport messages. Each function call is packed into a single IP packet and is answered by the server. Byte order is little endian (Intel-like). The default timeout is 60 seconds with only one function call allowed at a time. See the following two Tables 5 and 6 for an overview over the request-response protocol of EIBlib/IP.

The EIBlib/IP server should support the following bus access and configuration functions, listed with function IDs:

- Open (0001_h): The open function establishes a local EIB communication channel on the server, which afterwards is confirmed with a standard response telegram.

byte	0-3	4-7	8-11	12-...
description	message size	function identifier	version	parameter values

The request telegram is sent from the client to the server. The protocol version is currently fixed to 1.0 and is encoded as 0100_h.

Table 5: EIBlib/IP request telegram

byte	0-3	4-7	8-11	12-15	16-...
description	message size	function identifier	success indicator	error code	return values

The response telegram is sent from the server to the client after fully processing the request. The function identifier must match the one of the request. Success indicator and error code field give information about success or failure of a request. For a list of error codes refer to [62].

Table 6: EIBlib/IP response telegram

- Close (0002_h): This function shuts down the server-side EIB communication channel.
- Read (0003_h): The client uses this function – with the maximum number of requested bytes as a parameter – to read a telegram from the server. If one or more telegrams are available, the server should respond immediately with the oldest message in its queue. If no telegram becomes available within half of the IP timeout, the server should respond with an empty telegram.
- Write (0004_h): After opening and configuring an EIB communication channel, the client uses this function to send an EMI 1 telegram to the EIB. The server processes the packet immediately and responds with the actual number of bytes written.
- SetPar (0005_h): To configure a local EIB communication channel, this function is called by the client with the following parameters: PortName, BaudRate, DataBits, Parity, StopBits, BcuAnswerTimeOut, TelegramTimeOut, WriteIrpTimeOut and ReadTrpTimeOut. Every iETS server must return with success, otherwise a communication error occurs at client side.
- IsOpen (0006_h): The function checks for an open communication channel

at the server.

- Reset (0007_h): To reset the internal state of the iETS server, the client can call this function. No messages are transmitted to the EIB or the IP read channel.
- GetServerInfo (0008_h): Future versions of EIBlib/IP can use different frame formats than EMI 1. To get the supported EMI types, this function can be called. The server responds with one of the following two return values: 00000001_h for EMI 1 or 00000002_h for EMI 2.0 support.
- Identify (0009_h): The server responds to this request with its EIBA manufacturer code and its MAC address.

2.3.2 EIBnet/IP

EIBnet/IP describes transportation of KNX telegrams on top of IP networks, which are, due to their widespread deployment, an ideal fast backbone. The main purpose of EIBnet/IP is to expand building control beyond the local KNX bus. EIBnet/IP is transparent for KNX devices. Remote configuration and operation is possible via EIBnet/IP servers. Existing Internet Protocols (IPs) [58] are used, unless their implementation and memory usage requires huge efforts. Address Resolution Protocol (ARP) [55], Boot Protocol (BootP)/Dynamic Host Configuration Protocol (DHCP)[10] [19], User Datagram Protocol (UDP) [56], Internet Control Message Protocol (ICMP) [57] and Internet Group Management Protocol (IGMP) [15] are mandatory implementations whereas Reverse Address Resolution Protocol (RARP) or Transmission Control Protocol (TCP) [59] are optional. An EIBnet/IP server should be connected to an IP medium, being able to transmit at least twice the bit rate of all connected EIBnet/IP routers ([40] Part 03/08/01).

Until now, for KNX *security* was of minor concern, as any breach of security requires physical access to the network wires, which is nearly impossible in a building. When using an existing data network, however, several security threats need to be considered. These are, for example, eavesdropping, modification or deletion of messages and denial of service attacks. As EIBnet/IP does not provide any security measures, the underlying network has to be secured by, for instance, using virtual private networks, local intranet only or using authentication for opening tunnelling or remote logging connections. Refer to [32] for a detailed

[10] A server should support a fixed address, configured by ETS, as well as a dynamic address assignment.

discussion on this topic.

Currently there are four service protocols specified in EIBnet/IP. Every EIBnet/IP device needs to implement at least the basic *Core Service*. The core specification defines the packet structure. Moreover, it is responsible for discovery and self-description of an EIBnet/IP server and for configuring, establishing and maintaining a communication channel between the client and the server. In contrast to EIBlib/IP, EIBnet/IP uses big endian (Motorola-like) byte order.
Figure 7 explains the basic structure and function of the EIBnet/IP server and the core service. The server provides one well known discovery endpoint[11] to the IP network and one or more service containers. A server should at least support one control endpoint and one data endpoint[12] per KNX connection. If, for example, the gateway is connected to two different KNX networks, it provides two different service containers with both control and data endpoints. An endpoint can be uniquely addressed with the defined Host Protocol Address Information (HPAI) structure. HPAI is the necessary data to send EIBnet/IP frames to the communication partner. For IPv4 the HPAI currently contains the 4 byte IP address and 2 byte port number. For discovery of an EIBNET/IP server, the client sends a `SEARCH_REQUEST` with its own discovery HPAI to the system setup multicast address. Every server receiving the request should respond immediately with a `SEARCH_RESPONSE` frame for each of its service containers. It is addressed to the provided client HPAI and should contain the HPAI of the control endpoint. Afterwards, the client typically sends a `DESCRIPTION_REQUEST` to all received control endpoints using a unicast or point-to-point connection. Servers respond with a `DESCRIPTION_RESPONSE`, containing supported protocol, capabilities, state information and an optional friendly name.

A *communication channel* is the data endpoint connection of an EIBnet/IP client and an EIBnet/IP server. It is established by the client for services requiring a point-to-point connection like, for example, Tunnelling or Device Management. Before trying to establish a connection, the client should first check whether the requested mode is supported. It can then send a `CONNECT_REQUEST` frame to the control endpoint of the service container with its own data endpoint HPAI as payload. The server should respond within the connection request timeout

[11] The endpoint is fixed to UDP port 3671 and can be reached via the system setup multicast address 224.0.23.12.

[12] UDP as well as TCP on freely selectable ports are allowed for control and data endpoints.

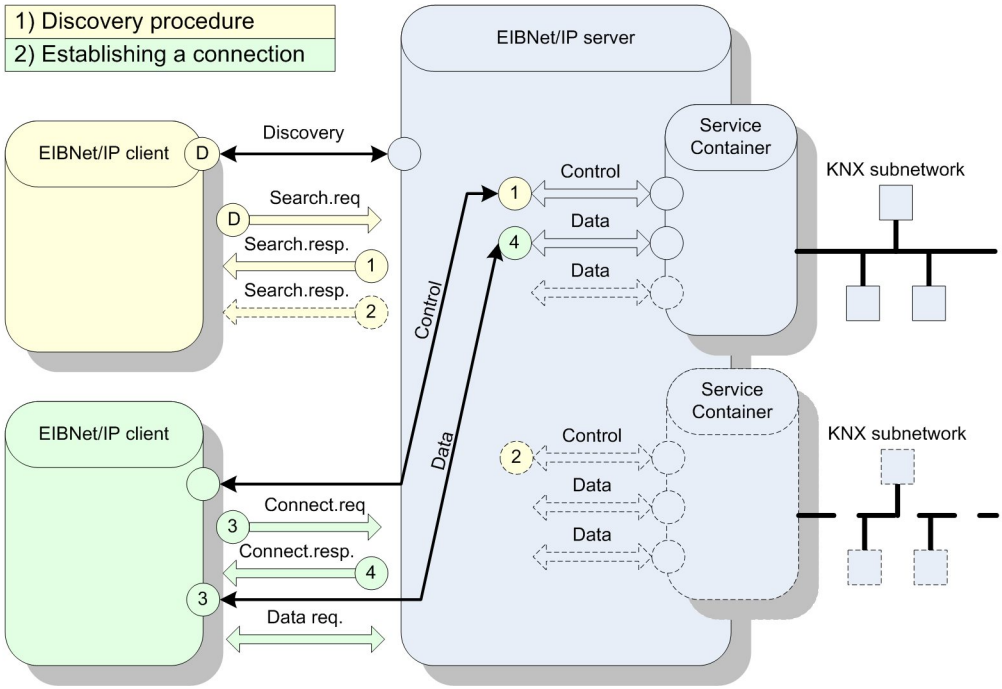


Figure 7: EIBnet/IP core function

with a `CONNECT_RESPONSE` containing the prepared data endpoint HPAI for this connection. Since an unreliable medium can be used for transporting EIBnet/IP frames, some sort of heartbeat monitoring and sequence counting must be provided by the protocol. For each established communication channel, a sequence number starting with '0' is maintained. For detection of communication failures, a heartbeat in form of a `CONNECTIONSTATE_REQUEST` is sent by the client every 60 seconds.

All EIBnet/IP telegrams consist of at least the common *EIBnet/IP header*. The first octet is the header size, which for the time being is fixed to 06_h. The second octet represents the EIBnet/IP version, which is currently fixed to 1.0. The following two octets describe the EIBnet/IP service type, according to which the telegram is passed to the destination layer. The upper octet denotes the service type family and the lower octet the actual service type of that family. The last two octets of the EIBnet/IP header are the sizes of the header and the message body. The following core EIBnet/IP services are defined:

- Search request: It is sent by an EIBnet/IP client via multicast to the discovery endpoints of any listening server. Since this connection is stateless, the client has to include its discovery HPAI in the body.

- Search response: The server responds with its control endpoints and a description of device hardware and supported service families.

- Description request: This frame is used to obtain a self-description of the EIBnet/IP server. It further contains the client's control HPAI, like the search request.

- Description response: The response is addressed at the provided return address (control HPAI) and contains various *Description Information Blocks (DIBs)* . The device information DIB provides information about the KNX medium, device status, physical and individual address, project-installation identifier, device serial number, routing multicast address, MAC address and device friendly name. The supported service families DIB lists, supported service type IDs and the manufacturer DIB identifies the device manufacturer and may contain additional data.

- Connect request: The connect request is sent by the client with its own control endpoint and desired data endpoint. Furthermore, it contains the additional *Connection Request Information (CRI)*, which for instance selects the requested connection type (management connection, tunnel connection, ...).

- Connect response: After successfully preparing the data endpoint, the server responds with the connection response telegram containing the communication channel ID, status information, data HPAI and a *Connection Response Data Block (CRD)*.

- Connection-state request: This frame is sent by the client to the control endpoint of the server. It contains the unique communication channel ID and the control HPAI.

- Connection-state response: The server responds with the communication channel ID as well as a status code (no error, ID not found, data connection error, KNX connection error).

- Disconnect request

- Disconnect response

See Figure 8 for the complete core frame format and services.

The second service type is EIBnet/IP *Device Management*, which defines management of EIBnet/IP devices. Configuration and management of the device can either be done via the IP network or the KNX network. In both cases the actual data is carried in cEMI telegrams. The use of IP allows larger data structures. The procedure is based on interface object properties: Each manageable value is assigned an unique property ID and a corresponding human readable property name. The client can send a `DEVICE_CONFIGURATION_REQUEST` packet containing the key and the actual value. The server processes the request and responds with a `DEVICE_CONFIGURATION_ACKNOWLEDGEMENT` within 10 seconds. Typical manageable values are, for instance, project installation ID, KNX individual address, IP address, or friendly name. The actual frame format can be seen in Figure 8. For a full list of property IDs refer to specification in ([40] Part 3/8/3).

The *Tunnelling* protocol describes the point-to-point exchange of KNX data over the IP network for configuration and diagnostics of devices on the KNX network. The tunnelling client sends cEMI management telegrams, contained in an IP telegram to the tunnelling server, which passes the data to the KNX network. All ETS functions are supported by tunnelling devices using this way. It has to be mentioned, that the protocol does not address timing issues caused by the IP network and therefore tunnelling protocol transfer time has to be smaller than the transfer timeout of KNX. There are three different tunnelling services in EIBnet/IP version 1.0:

- Tunnelling on KNX data link layer: Every EIBnet/IP device has to support this mode. Upon establishing a connection with a `CONNECT_REQUEST`, the EIBnet/IP server assigns a KNX individual address to this connection and passes it to the client in the `CONNECT_RESPONSE`. This distributed addresses can be configured and are stored in the property `PID-ADDITIONAL-INDIVIDUAL-ADDRESSES`. The EIBnet/IP server forwards all KNX point-to-multipoint (group addressing) telegrams contained in a `TUNNELLING_REQUEST` to the connected client, as well as the point-to-point telegrams, matching the assigned individual address. Furthermore, the server generates `IACK` frames (see Section 2.1.3) for the assumed addresses. If the tunnelling client sends a cEMI frame with KNX Source address set to 0000_h, the server replaces the address with the individual one

Figure 8: EIBnet/IP frame format

and passes the message to the KNX network. If the address in the cEMI frame is set, the server sends the telegram unmodified.

- Tunnelling in cEMI raw mode: In this mode, the server passes any KNX message received to the connected client. Moreover, it does not generate `IACK` frames. Implementation is optional.

- Tunnelling on KNX busmonitor: Activating KNX busmonitor mode may disable any other EIBnet/IP services for the KNX subnetwork. Hence Tunnelling on KNX busmonitor should not be supported in a EIBnet/IP Routing device. Implementation is optional.

For the EIBnet/IP Tunnelling frame format refer to Figure 8.

Routing is a point-to-multipoint protocol for routing messages between KNX devices over the high-speed IP network. EIBnet/IP routers send UDP/IP multicast messages to other EIBnet/IP routers on the same IP network, which in turn filter the messages according to their destination address or group address and eventually pass them to the KNX layer. EIBnet/IP routers can replace traditional KNX line and backbone couplers. Care has to be taken of assigning individual addresses to routers to ensure proper routing of KNX telegrams. ETS has therefore been improved with rules for address assignment to EIBnet/IP devices. To be able to receive the multicast packages from the other routers, IGMP is used. It informs the IP routers in the IP network of the desired IGMP membership of the EIBnet/IP router. If two different KNX installations are using the same IP backbone, they have to use different multicast addresses for their routers. To control the range (i.e. hops) a datagram can travel, routers can set the TTL to the required value. Since the IP backbone is a lot faster than the KNX network, overflows of the IP-to-KNX queue can occur because the router is not able to transmit all received frames to the KNX network. On this event, EIBnet/IP routers should maintain overflow counters and should transmit a `ROUTING_LOST_MESSAGE` to the IP network, allowing a central monitoring station to detect troubles in network design. The frame format can again be seen in Figure 8.

3 Requirements

In this section the requirements and foregoing efforts of how an appropriate platform should be realised are described. Without question, a lot of KNX/EIB platforms are already on the market and the question arises why to reinvent the wheel. To put it short, none of the existing solutions was able to meet all our constraints, which were the following:

- Universally applicable: The primary goal for our new platform is that it should be universally applicable as a gateway and interface for KNX/EIB. The main purpose is to serve as a basis for further work in the scope of home and building automation (e.g. plug and play facilities, integration into Open Services Gateway initiative (OSGi) environments [36], coupling to other networks, extensions with regard to security issues, setup of set-top boxes, access point for BASys [5]).

- Low cost and compact: The platform should be designed as a compact and low cost stand-alone device. This includes costs for PCB manufacturing as well as costs and availability of used components. Experienced users should be able to build it from scratch using cheap off-the-shelf components. Most important, no additional hardware or drivers should be needed.

- Experimental embedded platform: The platform will not be primarily designed for end-user resale. It is rather designed for lab use. This should not unnecessarily compromise later commercial use, however. For now, it should be possible to use it as an experimental platform, without, for example, proper housing.

- Robust: Despite the experimental use, the platform should be electrically safe and easy to handle. This includes, for example, galvanic isolation of KNX/EIB side and microcontroller side as well as a proper socket for the Secure Digital / Multimedia Card (SD/MMC).

- Flexible: To provide universal and flexible use, the platform should be configurable in hardware (by use of e.g. jumpers) and software. Furthermore, the hardware should be extensible: Future extensions should be possible. An interface which allows such functions should be provided.

- Ease of use: The platform should be easy to use, even for non-advanced users (e.g. provide a C compiler), but nevertheless powerful functions and

low level access should be possible. For testing purposes, all parts of the hardware should be accessible, so that, for instance, usage of an oscilloscope is possible.

- Sufficient resources: The used hardware should be powerful, meaning that enough processing power and memory should be present to implement services like e.g. TCP/IP or USB. Moreover, further extensions like cryptographic algorithms for secure KNX/EIB should not form an obstacle. For such applications some sort of persistent storage (>1MB) is required.

- Design openly available: The design of the hardware and software should be openly available.

3.1 Hardware

Hardware requirements of the platform are enumerated easily: Only an embedded design design is practical, as a PC-based design does not meet the above stated requirements. Important entries on the wish-list for the platform are an KNX/EIB connection, RS232 access as well as direct I/Os. Protocols like EIBnet/IP and KNX on USB are to be implemented and therefore hardware support for IP and USB must be present. Furthermore, additions like LON or Controller Area Network (CAN) [43] support should be possible. Selection of appropriate components depends on various factors. Main driving factor is an existing prototype gateway designed by Oliver Alt [4] and available know-how.

3.1.1 Microcontroller

A lot of different Micro Controller Units (MCUs) are present on the market. Selection of the model is not only based on supported features, but also on personal preferences. Hardware requirements are sufficient CPU power, RAM, flash memory and EEPROM, which preferably can be programmed directly by the MCU. At least a 16-bit CPU is required. Moreover, sufficient I/Os should be present. Due to the fact that no MCU with both integrated USB and Ethernet support exists up to now, a solution with an external bus interface is needed so the missing features can be connected externally. As mentioned earlier, in-system programming of the MCU should be possible (i.e. without external flashing equipment).

3.1.2 Ethernet controller

Connection to Ethernet can be realised in various ways. Solutions based on different interfaces exist: Connection to the CPU can performed via UART, Inter-IC (I^2C), Serial Peripheral Interface (SPI), Industry Standard Architecture (ISA) bus or Peripheral Component Interconnect (PCI) bus. Depending on the interface, different speeds (10 MBit/s, 100 MBit/s, ...) are possible and different implementation efforts have to be made. Controllers also differ in hardware support regarding supported layers: physical layer (layer 1) support must be present in all controllers whereas only some of them offer data link layer support (layer 2). The controller selected should provide as much hardware support as possible and for connection to the MCU application notes should exist.

3.1.3 KNX/EIB connection

Connection to the KNX/EIB bus can be realised using any of the TP1 interfaces presented in Section 2.1. The required passive components and PCB space should be as low as possible. Furthermore, low level access to the KNX/EIB bus should be possible to provide the basis for further academic work such as fault injection or traffic analysis. The platform also should provide the capability to act as a bus monitor and simultaneously be able to send frames to the KNX/EIB.

3.1.4 USB support

Integration of USB should be possible with minimum required effort and additional components. Ideally, the MCU has native USB support and drivers exist. For further extensions, support of the USB host feature would be a benefit.

3.1.5 Additional components

For permanently storing huge data amounts, some sort of persistent memory is required. An external medium with write and read support in standard PC devices like a compact flash or a secure digital card is desired. As standard serial connection, at least two UARTs with attached RS232 transceiver should be present.

3.2 Software

3.2.1 Development

Traditionally microcontrollers are programmed in assembler, but nowadays C compilers are usually shipped. The selected microcontroller should offer a free development environment with C compiler and debugging support. Moreover, application notes and programming samples should be available. Reference implementations should exist.

3.2.2 Implementation

Manifold possibilities arise if the hardware of the platform fulfils all desired requirements. It can not only form a platform for testing existing stacks and implementations, but also be integrated into a real KNX/EIB installation (via, e.g. the integrated web server). As a beginning, ETS support via EIBnet/IP should be possible. KNX on USB implementation is not performed by the author, but is currently under development. Furthermore, simple programs should be available, to test the underlying hardware. As further extension, BASys support should be integrated.

Architecture of the software should be modular and should support code reuse. Furthermore, it should be as hardware independent as possible.

4 Hardware

This section describes the design process – from selection of the components to realisation of the final hardware – of the platform.

The idea for a multipurpose EIBnet/IP gateway evolved in September 2004. Within the scope of a practical course, EIBnet/IP was implemented on a Fujitsu microcontroller. For development and testing purposes it was plugged into a Fujitsu development board. It featured a reset button and a bank of Dual In-line Package (DIP) switches for mode switching (e.g. flashing and operational mode) of the MCU. Two RS232 connectors were present, one being used for connection to Serial Line Internet Protocol (SLIP) and the other being used for connection to the TP-UART. The TP-UART and the required components were plugged into a circuit board [72]. It was connected to an UART of the MCU by a serial cable. The necessary drivers for the TP-UART were implemented in the firmware of the microcontroller. Connection to an IP network was realised over a SLIP [61] connection. For testing purposes, the experimental setup consisted of EIB/KNX sensors and switching actuators, as well as an Siemens IP Router (N146) that acted as counterpart for IP connections. For configuration and EIBnet/IP counterpart ETS was used. This setup allowed demonstration of the device management, tunnelling and routing capabilities of the gateway. Refer to [60] for a detailed description on the experimental gateway.

In March 2005 it was decided to design, manufacture and program an integrated standalone solution within the scope of this diploma thesis. In the beginning a lot of time had to be invested by the author to become familiar with hardware design and component selection. As project name “KNXcalibur” was chosen.

Section 4.1 depicts the approach, which emerges from the requirements mentioned in Section 3, and decision for the actual components. Section 4.2 outlines the hardware design of the final board and Section 4.3 contains important information regarding the operation of KNXcalibur.

4.1 Selection of components

Selection of components is primarily based on available documentation, application notes, software and firmware. The board is a further development of a proprietary EIB gateway developed by Oliver Alt. Especially miniaturisation is an important design goal.

4.1.1 Microcontroller

A microcontroller with integrated Ethernet module and USB module would fit best for KNXcalibur. However, no suitable, single chip and cheap device is present on the market. Various possibilities exist to connect external Ethernet chips to all sorts of MCUs (see Appendix A.1 for examples). USB support, however, is still rarely available and therefore a solution with native USB support is preferred. The MB90330 family of Fujitsu is one of the first microcontrollers with integrated USB device and USB mini host support. Connection to Ethernet can be realised with an existing concept [4] and therefore the controller has been selected for the platform. What is more, the author is already familiar with Fujitsu MCUs and therefore no additional initial effort is required to learn an unfamiliar technology. Good contacts to Fujitsu also exist[13].

- The MB90F334A microcontroller is a 16 bit controller of the 16-LX-family, manufactured in 0.35 μm technology. It has a maximum operating frequency of 24 MHz, which provides enough processing power.
- The controller features integrated 24 KB RAM and 384 KB flash ROM, which can be multiple programmed via a special PC program. The controller has enough memory to support simultaneous connections (e.g. multiple TCP connections) and multiple services like EIBNet/IP and a web server at the same time.
- 94 Input / Output (I/O) ports are present, whereof 72 are I/O pins and 22 open drain output pins. 16 I/O pins feature an internal pull up resistor programmable by software.
- The MCU has 4 UARTs, which are sufficient for two separate connections to the KNX/EIB, one programming connection and a debugging connection to the PC-side.
- The controller holds a lot of different timers: A 3 channel 16-bit reload timer, one 16-bit free run timer, output and input compare timers, 8/16 bit Pulse Pattern Generator (PPG) timers, and a 16-bit Pulse Width Count (PWC) timer.

[13]Thanks to Matthias Steeg of Fujitsu Microelectronics Europe GmbH for supplying the MB90F334A samples.

- The 8/10 bit Analog / Digital (A/D) converter is currently unused. It can be used for further extensions, such as a PEI connection with analog input pins.
- 8 external interrupt pins are present and are used to connect to the Ethernet controller as well as to the Save pins of the TP-UARTS.
- 3 channels for I^2C bus exist.
- The MB90330 features an external bus interface which is very similar to the ISA bus interface. With a little hardware support, ISA based Ethernet controllers can be connected. See Section 4.2.4 for details.
- The controller has one-channel USB support. USB functionality with device (2.0 full speed) and mini host support is integrated into the controller. Up to 5 endpoints can be used device mode.
- Fujitsu ships a free development environment with C compiler and debugging support. Furthermore, a software emulator is available. See [1] for details.

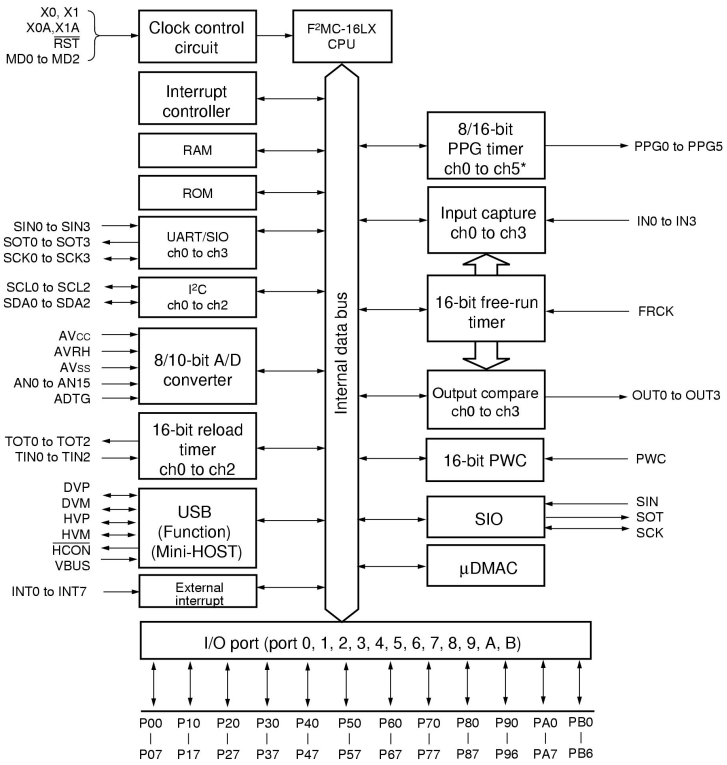


Figure 9: Block diagram MB90F334A

4.1.2 Ethernet controller

The IP connection can be realised in two ways. Firstly, there is the possibility to connect the platform over SLIP, being a protocol to transport IP packets over a serial line (see [61]). It is rather simple and was used for testing our prototype. Secondly, the platform can be connected using a dedicated Ethernet controller. See Appendix A.1 for Internet links to existing solutions. The Cirrus Logic CS8900A Ethernet LAN controller [10, 17] has been selected for use for the platform, due to an existing schematic [4] for the Fujitsu external bus interface. It is a single chip and low-cost controller for embedded applications, which does not need additional costly components. Furthermore, a comprehensive suite of software drivers is available. The CS8900A is available in a 100-pin Thin Quad Flat Pack (TQFP) package and has the following characteristics:

- The CS8900A supports a speed of 10 MBit, which is quite sufficient for a

platform, if the speed of KNX/EIB TP at 9600 bps is considered.

- The integrated RAM buffers transmit and receive frames and allows, in contrast to other Ethernet controllers, a design without additional memory.
- A direct ISA bus interface is provided by the controller, which allows easy connection to the MB90F334A. Via the ISA interface both direct memory access (memory mapped I/O) and indirect memory access (I/O mapped I/O) with I/O access and a description register are possible resulting in great flexibility concerning programming.
- Various important services are provided, so that no critical timings come up when transmitting or receiving frames. The controller offers automatic retransmission in case of collisions, padding as well as Cyclical Redundancy Check (CRC) computation.
- A lot of application notes and drivers for all operating systems exist, simplifying integration into KNXcalibur.
- The CS8900A should be very well available at local dealers or Internet distributors.

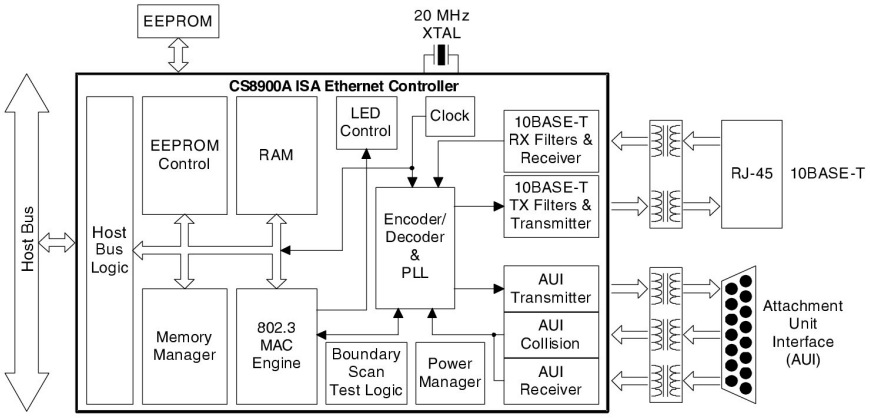


Figure 10: Block diagram CS8900A

4.1.3 KNX/EIB connection

Connection to KNX/EIB is realised with the TP-UART IC developed by Siemens [68]. It forms the easiest, most flexible and cheapest possibility to access the bus. The chip provides layer 1 and layer 2 access. No critical timings are present. Only the end of a telegram has to be detected, after 2 - 2.5 ms of bus silence, which does not form a barrier for the MB90F334A. Analysis mentioned in 3 may be performed using the analog mode of the TP-UART. The TP-UART is a 20 pin Small-Outline Integrated Circuit (SOIC) chip and requires little space on the PCB. Hardware schematic is supplied by the vendor. For galvanic isolation, optocouplers are used, which are supported natively by the chip (i.e. the transmit and receive pins of the TP-UART feature enough driving current). To connect the platform to different KNX/EIB TP lines and to provide the capability to act as a bus monitor and simultaneously be able to send frames to the KNX/EIB, two independent hardware parts (i.e. TP-UARTs) are needed which can be controlled individually.

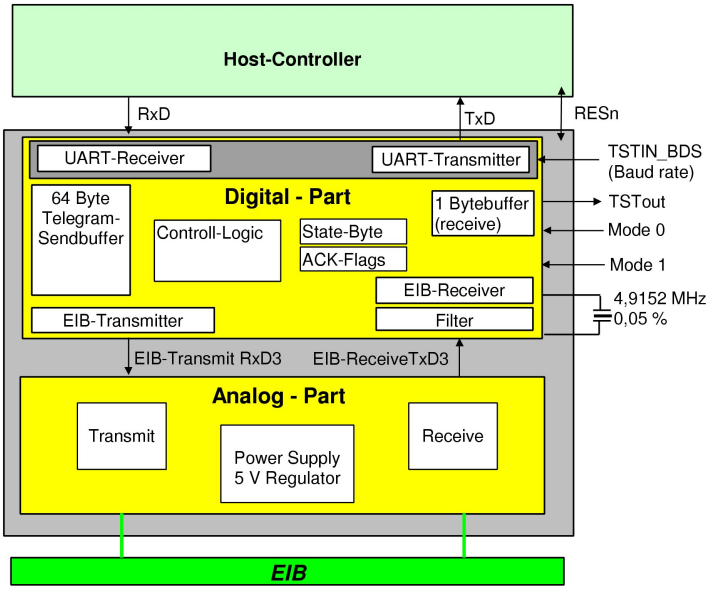


Figure 11: Block diagram TP-UART

4.2 Design

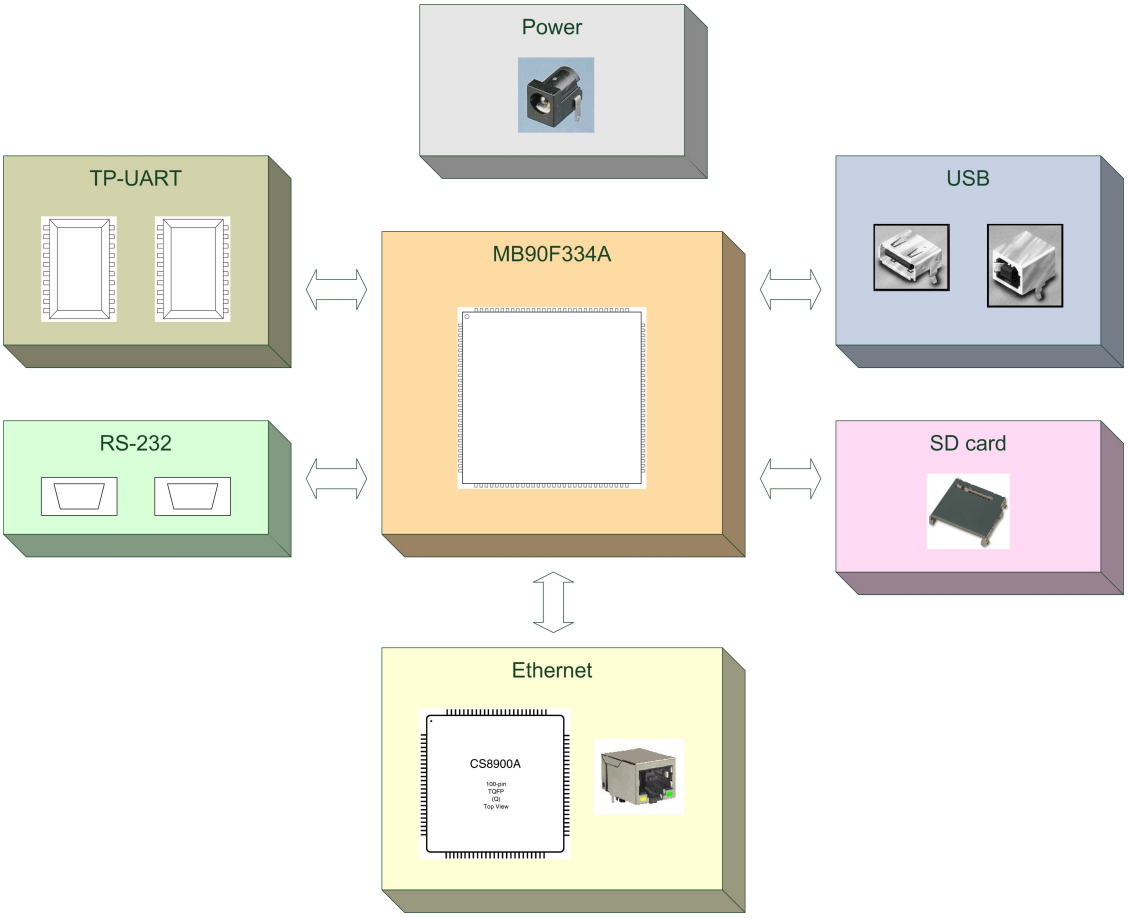


Figure 12: KNXcalibur: Hardware block diagram

The hardware design process of KNXcalibur consisted of various straightforward steps:

1. Initial thoughts: At the beginning, the structure of the platform was fixed. As shown in Figure 12 the following blocks exist: A power supply, two TP-UARTs, two RS232 connectors, a USB device and a USB host circuit and connectors, a SD/MMC card holder and an Ethernet controller and Registered Jack - 45 (RJ-45) connector.

2. Schematic: The schematic design evolved from application notes [31][14], [27, 25, 24, 28, 26, 10, 16], hardware manuals and data sheets [30, 29, 68, 17] and [4]. As layout editor CadSoft's Eagle [13, 14, 39] was selected. It offers sufficient possibilities and proved to be relatively easy to learn. See Appendix A.4 for the schematic[15].

3. Layout: After completing the schematic, the components were placed. The actual housing (pin count, package, ...) has to be considered at this time.

4. Routing: Eagle also supports routing of the circuit path. An autorouter is present, but does not produce the desired results. Therefore all tracks were routed manually. The component placement also was improved in this step to optimize the results. [35] served as a good tutorial about PCB design.

5. Manufacturing: The PCB has been manufactured by Beta LAYOUT GmbH - PCB-Pool [50]. A solder mask and position print were applied.

6. Mounting: Surface Mounted Design (SMD) as well as leaded components were used. SMD components with a pin pitch of 0.5 mm minimum proved to be difficult to handle in the beginning but finally they were soldered in successfully.

Figure 13 shows a partial rendering of KNXcalibur, created from the board design by Eagle3D (see A.1 for Internet link).

[14]Thanks to Peter Dörwald of Glyn GmbH & Co. KG for support.
[15]Thanks to Christian Kral for support.

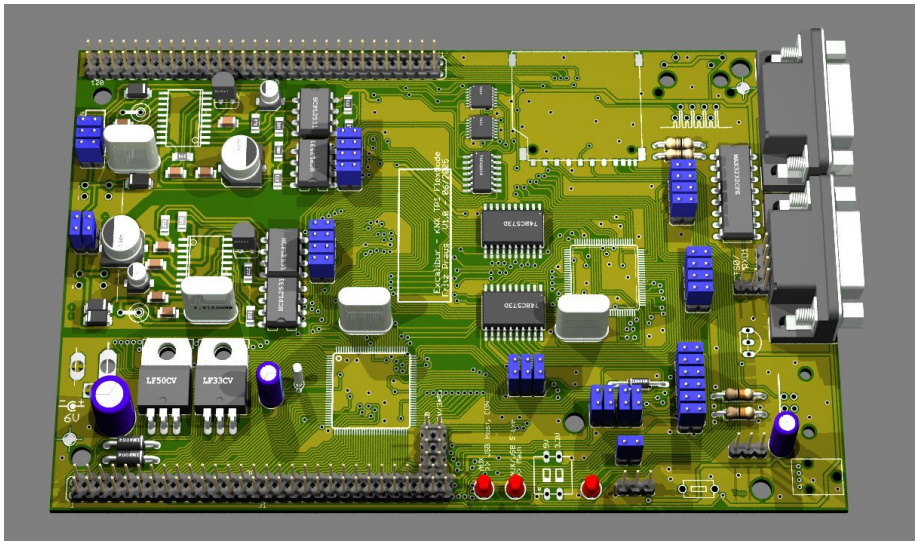


Figure 13: KNXcalibur: Eagle3D rendering

4.2.1 PCB design

The constraints of the PCB are derived from the requirements presented in Section 3. The board is in Eurocard format (160 mm $\times$ 100 mm) because this is a popular default size used in industry (and because Eagle is limited to this size in the standard version). It is double sided to keep down costs and simplify manufacturing. This implies that no dedicated ground or Voltage at the Common Collector (VCC) plane is present, resulting in a suboptimal solution regarding ElectroMagnetic Compatibility (EMC) design (see [25]). However, this trade-off has been accepted, most notably because a 2-layer design allows easier bug fixes. If errors would have been present in the schematic, traces could have been cut and fixed with wires. A 4-layer design with inner planes not accessible from outside does not allow that. To still provide measures against Electromagnetic Interference (EMI), a big ground plane on top and bottom layer was created. Furthermore, decoupling capacitors were installed for each connector pin to reduce the noise of external lines. Until now no troubles regarding EMI were encountered[16].

Signal tracks and gaps are at least 10 mil[17] wide. VCC lines are at least 24 mil

[16]The test environment is not particularly noisy regarding EMI.

[17]A mil is the imperial unit for $\frac{1}{1000}$ = 1 mili inch. In PCB design imperial and metric units are used. 100 mil (0.1 inch) are equal to 2.54 mm.

wide. Although track sizes and gap sizes of 8 or even 6 mil, which could be manufactured by PCB-POOL [50], would have simplified the routing process, a minimum width of 10 mil is set. So manufacturing technology is not exhausted and the board can also be manufactured by low priced, less technologically advanced companies. Using these track size constraints, no temperature rise of the tracks is noticeable and resistance, isolation and current capacity seem to be fine. For detailed physical tables about copper width, current capacity, resistance, isolation and temperature dependence see [76].

To achieve tighter integration and still be able to handle components easily, SMD parts of size 1206 (3.20 mm $\times$ 1.60 mm) and 0805 (2,00 mm $\times$ 1,25 mm) are used. Since not all parts are available in SMT, however, also leaded components are selected.

Two pin headers with 60 pins each are present on the board. They provide a one to one mapping of the pins of the MB90F334A (pin1 - pin60 = `CON1`, pin61 - pin120 = `CON2`). So extensibility is guaranteed and a daughter board on top of KNXcalibur can be realised. Moreover, they form ideal points for measurements (logic level, signal form, noise, ...) and bug fixes. An eight pin header is also assembled to provide power supply for the daughter board. Furthermore, all used peripherals can be connected or disconnected via jumpers (e.g. both TP-UARTs can be disconnected from the MCU). On the one hand this allows reuse of occupied pins and on the other hand they simplified the first startup of the platform. Errors can thus be located and eliminated easier. See Section 4.3 for a list of jumpers.

4.2.2 Power supply

The components of KNXcalibur require 3.3 V input voltage. 5 V power is only required for USB mini host connector supply and for optional extensions via a daughter board. Two low drop voltage (0.7 V) regulators [69] are used, which can be powered via an external mains adaptor or via the USB. Both regulators provide a maximum of 500 mA. For USB mini host 100 mA are sufficient (see [75]). Under load the platform currently requires about 160 mA. The power supply is therefore well dimensioned. The regulators are screwed down to the PCB to improve heat conductance. Nevertheless input voltage to the regulators should be as low as possible to keep power loss and heat dissipation low. To prevent short circuits – solder mask is not an isolation – under the regulators, a TO-220 isolator plate is mounted.

4.2.3 MB90F334A

The MB90F334A is fitted in a Low-profile Quad Flat Pack (LQFP)-120P package. Two different versions are available: FPT-120P-M05 with a pin pitch of 0.5 mm and FPT-120P-M21 with a pin pitch of 0.4 mm. In version 1.0 of the platform the M05 package is used. The MCU is directly soldered to the PCB. Use of a socket would increase costs significantly. Figure 14 shows the pin assignment of the MB90330 family. Look at Appendix A.2 and A.3 for pin description and memory map.

As minimum external circuit (see [27]) the MCU requires power supply, analog digital converter, clock, reset and mode-pins to be connected. As mentioned before, 3.3 V $\pm$ 0.3 V are required as power supply for normal mode as well as programming mode. 100 nF decoupling capacitors are placed at all supply pins of the on-board chips and SD/MMC socket. Two Light Emitting Diodes (LEDs) indicating the current state (on / off) are present. The analog/digital converter supply pins (`AVcc, AVss, AVRH`) are connected to VCC and ground via jumpers.

To the connectors `X0` and `X1` a crystal with 6 MHz resonant frequency and 22 pF bypass capacitors have to be connected. The maximum internal clock frequency of 24 MHz can be selected with PLL $\times$4 (see 5.1). Alternatively an external clock can be used. Furthermore a sub clock with 32.768 MHz at pins `X0A` and `X1A` also has to be connected.

A low active reset pin is present. It is pulled up with a 47 kΩ resistor. The MCU can be manually reset with a push-button.

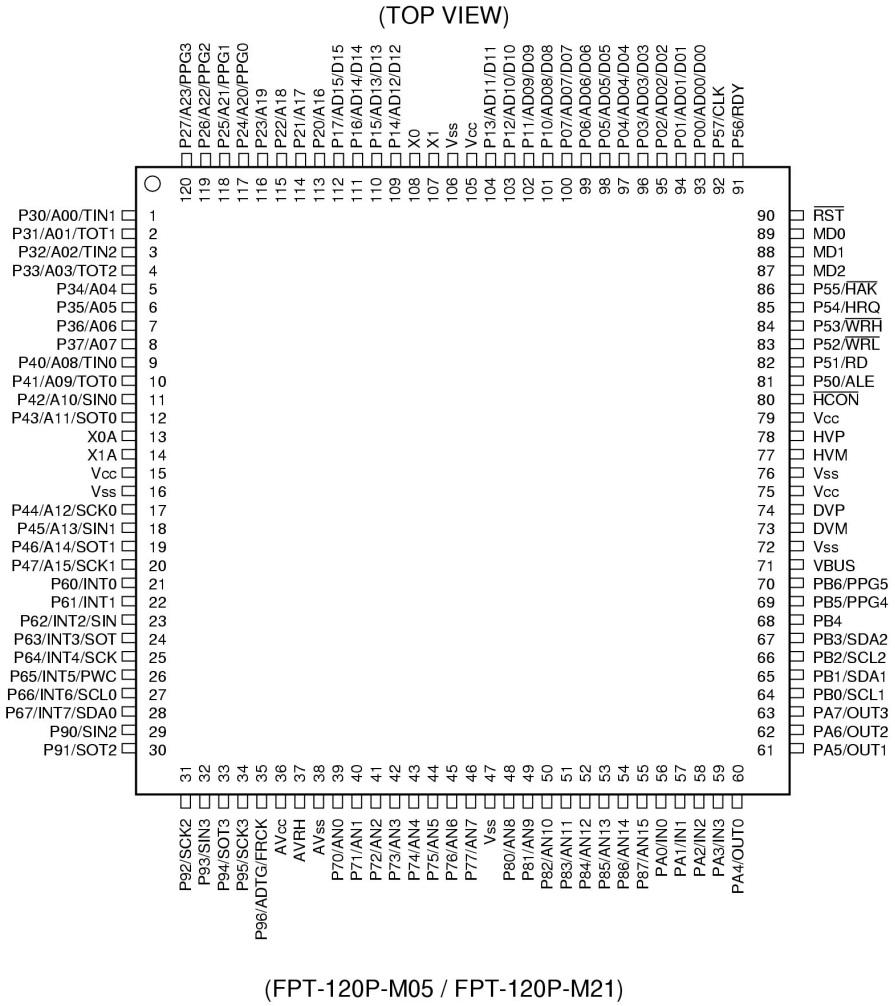


Figure 14: MB90330: Pin assignment

The MB90330 has three so called mode pins (`MD0`, `MD1`, `MD2`). They act as a switch for the current operation mode of the MCU. Six different modes are supported, which specify the load methods for the reset vector and the mode data. Two special pins (`P60`, `P61`) determine settings for programming mode. For a minimal system only two modes are necessary: *Flash Asynchronous Serial Programming Mode* and *Free Running Mode*. The former mode allows the firmware to be transferred from a PC to the MCU via the serial interface. The latter stands for normal operation. For easy mode switching a jumper has been placed, which

allows toggling of MD0/MD2 high or low level. Therefore, for switching between the two most important operating modes only a single jumper has to be moved. This is a considerable improvement over the Fujitsu board, where multiple DIP switches had to be set for this purpose. However, if other modes should be desired, JP6 has to be removed and the mode pins have to be set to GND via CON1. Furthermore, a LED has been installed, indicating the current mode. See Section 4.3 and Table 7 for details on usage.

P61	P60	MD2	MD1	MD0	Mode
x	x	0	1	1	Run
1	0	1	1	0	Flash

Table 7: MB90330: Mode pin settings

4.2.4 CS8900A

The CS8900A-CQ3 is housed in a LQFP-100P package with a pin pitch of 0.5 mm. As is the MCU, it is directly soldered to the PCB. The connection to the MCU has been adapted from Oliver Alt's[18] EIB gateway. This applies especially to the ISA address decoder logic.

The external bus interface of the MCU uses multiplexed address- and data lines at pins AD00-AD15 and A16-A23 (see Figure 14). The ISA bus, however, uses separate address and data lines. With the help of two address latches of type 74LVC573AD [54] and the ALE (address latch enable) signal of the MCU the two signals can be separated again.

The ISA architecture distinguishes in hardware between I/O access to peripheral devices (I/O mapped I/O) and memory accesses. In such an architecture the programming model distinguishes between I/O acesses through special commands and memory accesses. In the 16-LX-family, however, I/O accesses are mapped into memory area (memory mapped I/O). To support both access kinds of the ISA bus, a special logical circuit is needed that creates/emulates the four I/O-signals of the ISA bus (IOREAD, IOWRITE, MEMREAD, MEMWRITE). The logic circuit consists of three logic gates (two OR gates and one inverter). It has to be noted that the ISA bus uses negative logic for control signals.

[18] Big thanks to Oliver Alt for support and providing schematics and software.

The whole ISA bus area is mapped into the address space of the MCU. Since the ISA bus uses only 20 address lines (20 bit) and the MCU has 24 address lines (24 bit), the upper four lines of the MCU can be used to select between memory and I/O area. As seen in Figure 15 the address bit `A20` of the MCU is defined as the control signal. Depending on its value, I/O or memory mode is selected and the additional logic creates the read/write signals. If `A20` is low (I/O access), all I/O addresses are selected whereas if it is high, a memory access occurs. Thus the address space 200000_h - $2FFFFF_h$ is used for I/O and the address area 100000_h - $1FFFFF_h$ for memory access. `A22` is used as a chip select bit. If it is high, the whole decoder logic is out of function and the attached CS8900 is not addressed. To give an example: If the MB90F334A generates the address 200300_h, the address seen by the CS8900A will be 00300_h with one of its I/O commands (`IOR` or `IOW`) active. Similarly when the MB90F334A generates address 101400_h, the address seen by the CS8900A will be 01400_h with one of its memory commands (`MEMR`, `MEMW`) active.

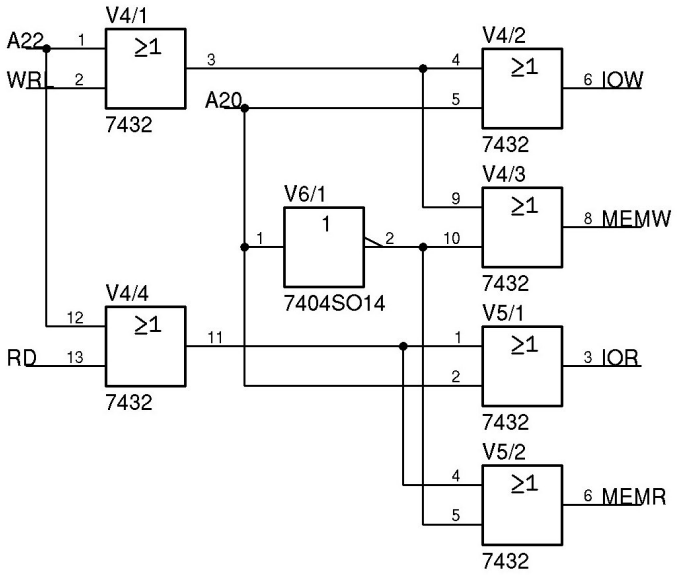


Figure 15: ISA address decoder logic

Timing has to be considered when connecting the MB90F334A and the CS8900A. For correct operation, fast logic gates have to be used. The 74LVC04D [52] and 74LVC32APW [53] families were chosen. According to [17] the CS8900A requires roughly 135 ns at maximum for a 16-bit I/O read. For this

reason wait cycles for accesses to the external memory area have to inserted by the MB90F334A. This can easily be done by altering the initial start.asm file. At an operating frequency of 24 MHz the minimum execution time of an instruction is 41.6 ns. Therefore three wait cycles should be sufficient and are defined in the start.asm. See Section 5.1 for details.

Their propagation delays are sufficiently low. A further adaption is needed for the reset signal of the CS8900A. In the ISA architecture it is high active whereas for the MCU it is low active. The interrupt outputs of the CS8900A are connected to the interrupt inputs of the MCU. A jumper allows selection between the four interrupt pins. Only one interrupt pin has to be connected. Connection to Ethernet is realised with a RJ-45 connector with integrated transceiver and LEDs [73][19]. Routing of the address and data lines is not quite optimal due to the 2-layer constraint and the required pin headers. Nevertheless, no parasitic effects have occurred or have been measurable yet.

4.2.5 RS232

Serial connection to PC side has been realised with `UART0` and `UART1`. `UART0` is primarily used for flashing whereas `UART1` is freely available and currently used for debugging. A MAX3232 [45] with 2×2 channels is mounted as RS232 converter with true level and SUBD connectors are placed. Via jumpers either the CTS and RTS line of connector 0 or the transmit and receive lines of connector 1 can be connected to the MAX3232. This allows, for instance, the PEI 16 protocol to be implemented. See Section 4.3 for details.

4.2.6 TP-UART

To support bus monitoring as well as normal operation mode on the KNX side, two TP-UARTs are connected. They are connected galvanically isolated to the MCU via optocouplers [3] and their circuit has been routed single-sided on the top layer to provide a clear boundary between the TP-UARTs and other components. The circuits on TP-UART side are powered via the KNX/EIB network. By setting a jumper, `TPUART1` can be operated in analog mode. Moreover, the two TP-UARTs can be connected to different KNX networks, enabling the device to act as a coupler. `TPUART1` is connected to `UART3` of the MCU, `TPUART2` is connected to `UART4`. For hardware resetting, two reset lines are also connected to I/O pins

[19] Thanks to Christina Jackob of UMEC elektronische Komponenten GmbH for supplying the UE-LT1S023A-34 RJ-45 sample connectors.

of the MB90330. The TP-UART features a save pin, indicating a break-down of bus voltage for more than typically 1.5 ms. These pins are connected to the external interrupt pins `INT4` and `INT5` of the MCU and can also be polled. If bus power is present, a logical ‘1’ can be read whereas a logical ‘0’ is read if there is no power.

4.2.7 USB connection

USB hardware setup has been implemented as described in [31] and [74]. A USB-B and an USB-A connector [47] are present on KNXcalibur. With the USB-B connector the board can be connected to a PC (USB-Host). The board then functions as an USB device and no external power supply needs to be connected. Simultaneous use of the Mini-Host feature is not possible. Via a jumper the USB attachment procedure can be configured. The controller can either always register to the USB after attachment or be configured to only register via software. To connect devices (e.g. keyboards) to the platform the USB-A connector is used. KNXcalibur then functions as Mini-Host. External power supply is needed in this case.

4.2.8 SD/MMC card connection

To support persistent data storage without requiring writing to the MCU on-chip flash memory and to extend available memory, a SD/MMC card connection has been integrated. Only minimal additional hardware is required, which was the main requirement. Moreover, the SD/MMC card is “smart” and can be written and read in normal card readers. The SD/MMC card can be accessed via hardware or software SPI. Since the hardware SPI pins are shared with the interrupt pins, which are used for TP-UART connection, the SD/MMC card holder is connected to standard I/O pins and SPI is emulated. At least four signals (`CS`, `CMD/DI`, `CLK/SCLK`, `DAT/DO`) have to be connected. The card holder also has two low active pins for detection of card and write protection. They are connected too. In version 1.0 of KNXcalibur the two required pull-up resistors were forgotten and so always a ‘0’ was read from the pins. Two provisional pull-up resistors have been soldered in to fix this bug. For description of SD/MMC see [66] and for bus timing constraints see [67].

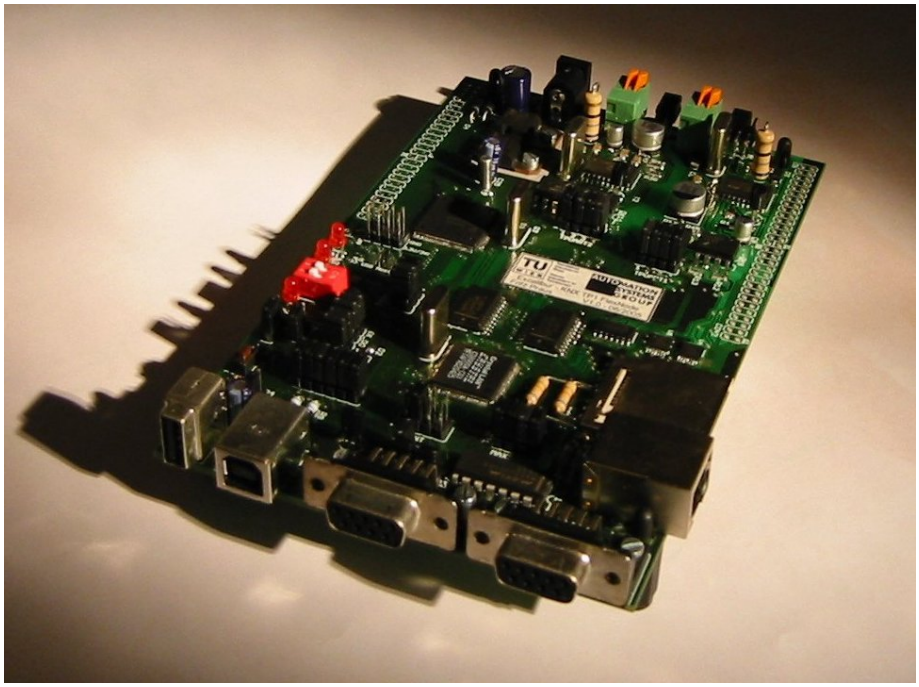


Figure 16: KNXcalibur: Picture

4.3 Usage

For safely operating KNXcalibur, at least this Section and Section 5.1 have to be read and understood. Under normal circumstances it should be impossible to physically damage the hardware by software. Care has to be taken when pins are forced to a specific level by hardware (e.g. `P60` and `P61`) and not to configure them as output by software. When attaching own peripherals via, for instance, `CON1` or `CON2` schematics and Section 4 have to be considered to avoid toasting the platform.

As mentioned earlier, KNXcalibur has no proper housing. Hence power supply lines and EIB lines are reachable. Of course, the applied solder mask provides some sort of electric isolation but at least the connectors/pins, pads and vias of resistors or chips lie freely. Care has to be taken to not get an electric shock or produce a short circuit. In version 1.0 of KNXcalibur the drill holes of the USB connectors are not tinned and soldered in and hence the connectors are not properly mounted. Avoid plugging cables without properly fixing the connectors.

To use the platform, obviously a power supply is needed. The first possibility

is to use the power supplied by a USB host. Only 3.3 V operation is possible and hence the USB Mini Host function and 5 V part of the AUX connector cannot be used. `Power switch 1` of `S2` and the 5 V AUX power led `LED2` have to be off. `Power switch 2` of `S2` can be used to turn the platform off or on. `LED3` indicates the current state. If the USB Mini Host is needed, an external power supply via connector `X3` must be connected. 6 V - 16 V can theoretically be used. The two voltage controllers, however, are cooled passively and in order not to heat them up too much, 6 V power supply is strictly recommended.

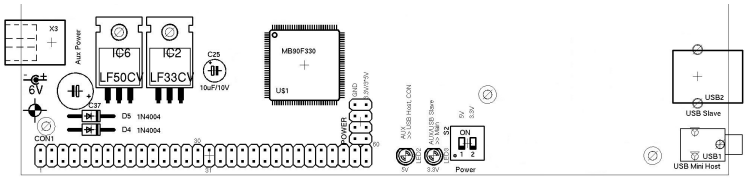


Figure 17: KNXcalibur: Power supply

The platform offers a DIP button `S1` to reset the platform. As mentioned before, the MCU has various operating modes. Power has to be off to switch modes. The flash mode is used to program KNXcalibur via `UART0`: Jumper `JP1` has to be closed and a PC has to be connected via a standard serial cable. Furthermore, `JP8` and `JP10-1` have to be closed and `JP10-2` has to be open. `JP6` has to be in state `1+2` and the programming LED `LED1` is automatically turned on. To select run mode, `JP6` has to be in state `2+3`, programming LED being off. To switch operating mode as long as no additional hardware is connected via `CON1`/`CON2` and data direction settings are considered, all other jumpers can remain unchanged. See Figure 18 and Table 8 for details about available jumpers and functions.

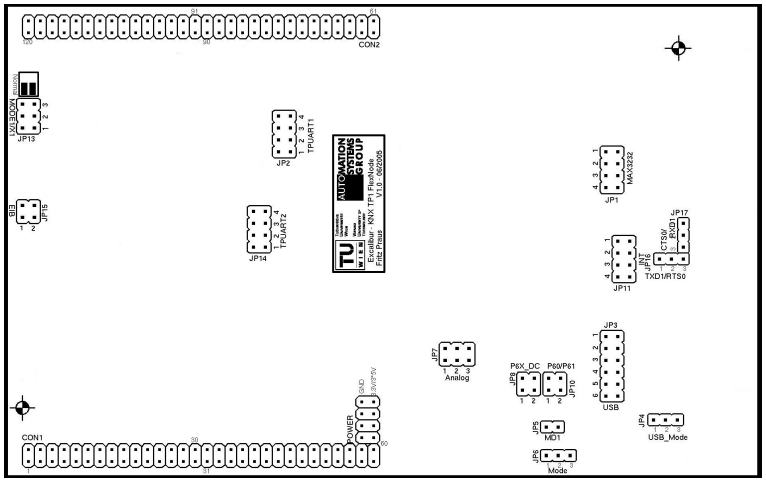


Figure 18: KNXcalibur: Jumper location

As the platform has been designed to be extensible, all peripherals can be disconnected or connected and configured via jumpers. Moreover, not all pins of the microcontroller are set to a specific level by hardware (by e.g. pull-down resistors). To prevent floating and possible physical damage of the hardware, the corresponding pins have to be set to a specific logic level by software. This can be achieved with the help of the port data direction registers of the MB90F334A. Setting them to output and writing a value prevents floating of unconnected pins. This step should be performed at the beginning of the user application. Nevertheless at least JP7 has to be closed to prevent the analog digital converter pins from floating. To give an overview of connected and used pins and their data direction refer to Table 9. For usage of KNXcalibur, reading Table 9 in the following way may be useful: For example, to determine the connection of the data input pin of the SD/MMC card, look at columns “Port” (“3”) and column “Name” (“1”). The DI signal is connected to hardware pin 1 of the MCU, which can be accessed via the synonym to “P31”.

Name	Type	Value
S2	Power	power switch (1=5V, 2=3.3V)
CON1	Pinheader	one to one mapping of pins 1-60 of MB90F334A
CON2	Pinheader	one to one mapping of pins 61-120 of MB90F334A
POWER	Pinheader	power to external peripheral (pin1=3.3V, pin2+4+6+8=GND, pin3+5+7=5V)
JP1	MAX3232	dis/connect MAX3232⇔MB90F334A
JP2	TPUART1	dis/connect TPUART1⇔MB90F334A
JP3	USB	dis/connect USB⇔MB90F334A
JP4	USB_Mode	USB logon (1+2=pull up, 2+3=transistor and HCONX)
JP5	MD1	mode pin 1 value (open=high, closed=low)
JP6	Mode	easy mode selection, observe programming LED LED1 (1+2=programming mode, 2+3=run mode)
JP7	Analog	set level of analog pins to prevent floating (ADC input)
JP8	P6X_DC	dis/connect JP10⇔MB90F334A to freely use P60/P61
JP10	P60/P61	set pin60 and pin61 to defined level (JP10-1 closed and JP10-2 open for flashing) (open=high, closed=low)
JP11	INT	dis/connect interrupt pins CS8900A⇔MB90F334A (1+2=INT0, 3+4=INT1, 5+6=INT2, 7+8=INT3)
JP13	MODE1/X1	set operating mode TP-UART1 (2x 1+2=normal mode, 2x 2+3=analog mode)
JP14	TPUART2	dis/connect TPUART2⇔MB90F334A
JP15	EIB	dis/connect EIB/KNX0⇔EIB/KNX1 (1+1 and 2+2=open/closed)
JP16	TXD1/RTS0	dis/connect UART0/1⇔MB90F334A (1+2=TXD1, 2+3=RTS0)
JP17	CTS0/RXD1	dis/connect UART0/1⇔MB90F334A (1+2=RXD1, 2+3=CTS0)
JP16/JP17	RTS0/CTS0	dis/connect RTS0⇔CTS0 (JP16-3 + JP17-3=RTS0/CTS0 open/closed)

Table 8: KNXcalibur: Jumper settings

Peripheral	Port	Bit	HW Pin	Direction	Func
TP-UART1	9	0	29	in	SIN2
		1	30	out	SOT2
		2	31	out	RES2
	6	4	25	in	INT4
TP-UART2	9	3	32	in	SIN3
		4	33	out	SOT3
		5	34	out	RES3
	6	5	26	in	INT5
CS8900A	0	0-7	93-100	in/out	AD
	1	0-7	101-104,109-112	in/out	AD
	2	0-7	113-120	in/out	AD
	6	0-3	21-24	in	INT0-3
	5	0-7	81-86,91,92	in/out	BUS
P60/P61	6	0,1	21,22	in	MODE
UART0	4	2	11	in	SIN0
		3	12	out	SOT0
UART1	4	5	18	in	SIN1
		6	19	out	SOT1
SD/MMC	3	0	1	out	CS
		1	2	out	DI
		2	3	out	CLK
		3	4	in	DO
		4	5	in	CD
		5	6	in	WP
unused	3	6,7	7,8	/	/
	4	0,1,4,7	9,10,17,20	/	/
	6	64-67	25-28	/	/
	7	0-7	39-46	/	/
	8	0-7	48-55	/	/
	9	6	35	/	/
	A	0-7	56-63	/	/
	B	0-7	64-70	/	/

Table 9: KNXcalibur: Port connections

5 Software

This section is split into three main parts. The first Section 5.1 describes the necessary steps to flash the microcontroller and to load one's own application to the MCU. Furthermore, the most important development tools are explained and the required hardware code constraints are listed. Section 5.2 describes available low level firmware from an abstract and general point of view. For detailed documentation of available low level functions (e.g. `initUART()`, `printfUART()`, ...) see comments in the source files. The available tools to test the hardware are also documented. The last Section 5.3 describes the available stacks/applications, which are partly or fully implemented.

5.1 Initialisation and usage

5.1.1 Development tools

Fujitsu ships various freely available and non-restrictive[20] development tools with their microcontrollers. `Softune workbench` is an IDE for the Windows operating system (see Appendix A.1 for Internet link) with integrated C compiler – which can also be used standalone – and simulator. It also features an integrated project management function. Programming can either be done in C or assembler. All KNXcalibur related software is written in C and has been integrated into a common workspace[21] managed by `Softune`. It consists of a common lib and different projects. To create a new project for an application, it is best to copy an existing project and adapt filenames. This way all special settings (assembler, compiler, linker, ...) for the MB90F334A are maintained and the user does not have to worry about them. Only the reference to the common library has to be adapted. Application notes for various functions can be downloaded from the Fujitsu homepage (See Appendix A.1 for further Internet links).

[20] e.g. without runtime licences

[21] Obtain sources from project homepage (see Appendix A.1) and open `fpraus/MB90330/KNXcalibur/KNXcalibur.wsp`.

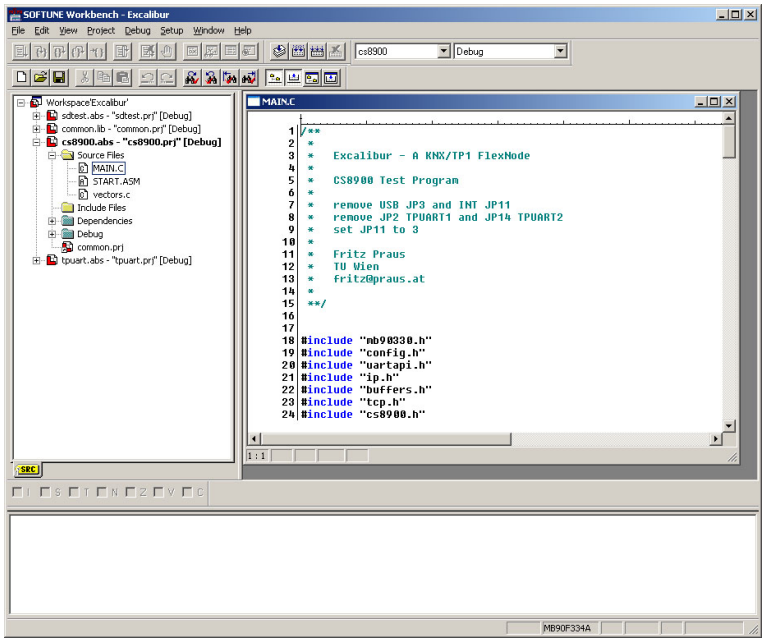


Figure 19: Fujitsu Softune Workbench

A downloadable hex file is generated via the build button in Softune. The `*.mhx` is located in the `abs` subdirectory of the project by default. It can be downloaded with the supplied flash programmer. Necessary hardware settings on the KNXcalibur board can be seen in Section 4.3. For KNXcalibur an external frequency of 6 MHz and the controller MB90F334 have to be selected. See Figure 20 for an example.

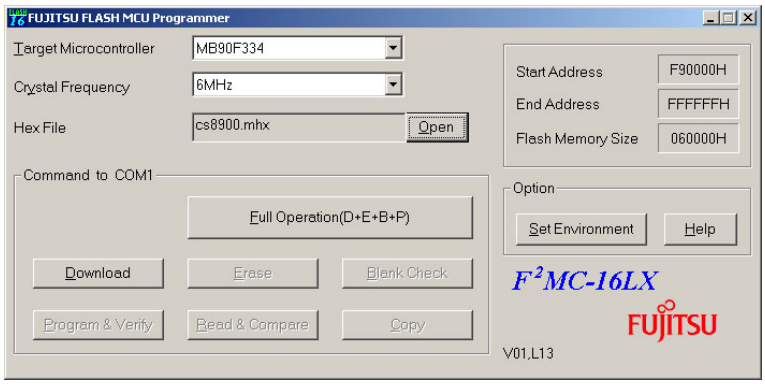


Figure 20: Fujitsu MCU Flash Programmer

As mentioned earlier, `UART1` is currently used for advanced debugging messages. Counterpart on PC side is any type of terminal. The integrated terminal function of Windows called `Hyperterminal` can be used. The provided functions are quite sufficient for first tests and implementations.

5.1.2 Usage

To initialise the microcontroller and additional hardware, several steps are necessary. These steps are project dependent and hence the corresponding files are located in the `Src` subdirectory of the project.

At first configurations, like operating frequency, external bus mode, ... are done by the *start.asm* assembler file, which is provided by Fujitsu and intended to be customised by the user. Although it is not necessary to use this file, it is strongly recommended. Structure of the file is as following:

```
;===================================================================
; 1  Contents
;===================================================================
; 1   Contents
; 2   Disclaimer
; 3   History
;
; 4   SETTINGS              (USER INTERFACE)
; 4.1   Controller Family
; 4.2   Memory model
; 4.3   Constant Data Handling
; 4.4   Stack Type and Stack Size
; 4.5   General Register Bank
; 4.6   Low-Level Library Interface
; 4.7   Clock Selection
; 4.8   External Bus Interface
; 4.9   Reset Vector
; 4.10  Enable RAMCODE Copying
; 4.11  Enable information stamps in ROM
;
; 5   Section and Data Declaration
; 5.1   Several fixed addresses (fixed for MB90xxx controllers)
; 5.2   Declaration of __near addressed data sections
; 5.3   Declaration of RAMCODE section and labels
; 5.4   Declaration of sections containing other sections description
; 5.5   Stack area and stack top definition
; 5.6   Direct page register dummy label definition
;
; 6   Start-Up Code
; 6.1   Import external symbols
; 6.2   Program start (the reset vector should point here)
; 6.3   "NOT RESET YET" WARNING
; 6.4   Initialisation of processor status
; 6.5   Set clock ratio
; 6.6   Set external bus configuration
; 6.7   Copy initial values to data areas.
```

```
; 6.8   Clear uninitialised data areas to zero
; 6.9   Prepare stacks and set the active stack type
; 6.10  Set Data Bank Register (DTB) and Direct Page Register (DPR)
; 6.11  Wait for PLL to stabilise
; 6.12  Initialise Low-Level Library Interface
; 6.13  Call C-language main function
; 6.14  Shut down library
; 6.15  Program end loop
; 6.16  Configuration stamp in ROM (currently test only)
; 6.17  Debug address specification
```

The user has to adapt the definitions in part 4.

- 4.1 Controller Family: The MB90F334A is part of the MB90300 series.
- 4.2 Memory model: This settings determines the address width of the `callp` function call for the main routine. Setting of the memory model[22] also needs to be done separately for the C compiler. Leave at `AUTOCONST`.
- 4.3 Constant Data Handling: Constants can be stored in RAM or ROM. Since ROM-mirror, which mirrors the flash memory into a 16-bit addressable memory space, is available, constants should be stored in ROM via `ROMCONST`.
- 4.4 Stack Type and Stack Size: The MCU supports two stack pointers. They are useful for operating systems. For the current implementation only the `SYSSTACK` pointer is relevant.
- 4.5 General Register Bank: Currently register bank 0 is sufficient. See [30] for description.
- 4.6 Low-Level Library Interface: To use C functions like `printf`, several low-level routines need to be defined and this option `CLIBINIT` needs to be on. Leave at off for current firmware.
- 4.7 Clock Selection: With the help of the internal PLL the operating frequency can be selected. $\frac{1}{2}$, 1, 2, 3, 4 are possible, 4 setting the maximum frequency of $6 \times 4 = 24$ MHz.
- 4.8 External Bus Interface: The external bus interface can be configured freely. For use with the CS8900A and decoder logic, `INTROM_EXTBUS`,

[22] The MCU supports different types of memory accesses due to speed and code size differences: 16-bit accesses are faster and smaller than 24-bit accesses. The modes `SMALL`, `MEDIUM`, `COMPACT` and `LARGE`, which alter address size of code and data, can be selected.

ADDRESSMODE MULTIPLEXED and three wait states for access need to be defined.

- 4.9 Reset Vector: The reset vector is not hardwired. Set to on to generate a vector.
- 4.10 Enable RAMCODE Copying: The MCU supports executing code from RAM. Leave at off.
- 4.11 Enable information stamps in ROM: Set to off.

The second important file is *vectors.c*. The Interrupt Service Routines (ISRs) and interrupt levels are defined here. 8 different levels, 0 being the highest, can be defined. They are assigned to the Interrupt Control Registers (ICRs), which each have two interrupt sources. To put it differently: Two interrupts share a single priority. Here is an extract from the file:

```
void InitIrqLevels(void)
{
/*  ICRxx               shared IRQs for ICR */

    ICR00 = 7;      /*  IRQ11
                        IRQ12 */
    ICR01 = 7;      /*  IRQ13
                        IRQ14 */

.
.
.

/*---------------------------------------------------------------------------
   Prototypes
   Add your own prototypes here. Each vector definition needs is proto-
   type. Either do it here or include a header file containing them.
---------------------------------------------------------------------------*/

__interrupt void DefaultIRQHandler (void);
__interrupt void rcUART1(void);

.
.
.

#pragma intvect DefaultIRQHandler 38    /* Ext. SIO                    */
#pragma intvect rcUART1           39    /* UART 0 RX / UART 1 RX       */

.
.
.

__interrupt
void DefaultIRQHandler (void)
```

```
{
    __DI();                             /* disable interrupts */
    while(1)
        __wait_nop();                   /* halt system */
}
```

The keyword `__interrupt` denotes a function as interrupt function. Interrupt nesting is also possible and can be circumvented by globally disabling interrupts via `__DI()`.

The hardware components (UARTs, timers, ...) of the MB90F334A are initialised and controlled via registers, which are mapped to the memory space of the MCU. Access to these memory spaces is simplified by definitions provided by Fujitsu. The *mb90330.h and mb90330.asm* map I/O memory spaces to the defined names in the handbook. Furthermore, all available bits are defined. So register assign operations look like `PDR = 8` and bitwise operations look like `PDR_P00 = 1`.

As mentioned in Section 4.2, not all pins are set to a specific level by hardware. To prevent floating pins, unused pins have to be set to output. Several routines have been developed to assist the user in configuring the *data direction registers*. They are defined in the file `config.c` and need to be called by the user right at the beginning of application code (i.e. first lines of `main()`). At least `initBoard()` needs to be called if all jumpers are disconnected. If, however, a specific hardware part of the board is connected – the jumper being closed –, then the corresponding function has to be called to prevent driving of logic levels on lines against each other.

```
/**
 * set data direction ports to output
 * and write 0
 * to prevent floating pins
 **/
#pragma inline initBoard
void initBoard(void) {
//  DDR0 = 0xFF; /* do not touch address      */
//  PDR0 = 0x00; /* and data lines of         */
//  DDR1 = 0xFF; /* the external businterface */
//  PDR1 = 0x00; /* already defined by 4.8    */
//  DDR2 = 0xFF; /* in start.asm              */
//  PDR2 = 0x00;
  DDR3 = 0xFF;
  PDR3 = 0x00;
  DDR4 = 0xFF;
  PDR4 = 0x00;
//  DDR5 = 0xFF; /* do not touch control lines   */
//  PDR5 = 0x00; /* of the external businterface */
  DDR6 = 0xFC; /* do not set P60+P61 to output */
  PDR6 = 0x00; /* they are set to ground by jumpers for flashing mode */
               /* remove jumper JP8 if you want to use these pins     */
```

```
  DDR7 = 0xFF;
  PDR7 = 0x00;
  DDR8 = 0xFF;
  PDR8 = 0x00;
  DDR9 = 0xFF;
  PDR9 = 0x00;
  DDRA = 0xFF;
  PDRA = 0x00;
  DDRB = 0xFF;
  PDRB = 0x00;
}

#pragma inline initP6X
void initP6X(void); // currently not used

#pragma inline initUART0
void initUART0(void) {
  DDR4_D42 = 0; // SIN0 input
}

#pragma inline initUART1
void initUART1(void) {
  DDR4_D45 = 0; // SIN1 input
}

#pragma inline initKNX0
void initKNX0(void) {
  DDR6_D64 = 0; // set INT4 input
  DDR9_D92 = 1; // Reset output
  DDR9_D90 = 0; // SIN2 input
}

#pragma inline initKNX1
void initKNX1(void) {
  DDR6_D65 = 0; // set INT5 input
  DDR9_D95 = 1; // Reset output
  DDR9_D93 = 0; // SIN3 input
}

#pragma inline initSD
void initSD(void) {
  DDR3_D33 = 0; //set pin MMC_DI to Input
  DDR3_D32 = 1; //set pin MMC_Clock to Output
  DDR3_D31 = 1; //set pin MMC_DO to Output
  DDR3_D30 = 1; //set pin MMC_Chip_Select to Output
  DDR3_D34 = 0; //set pin MMC_CI to Input
  DDR3_D35 = 0; //set pin MMC_WP to Input
}

#pragma inline initETH10
void initETH10(void) {
  DDR6_D62 = 0; // set INT2 input
}

#pragma inline initUSBHost
void initUSBHost(void); // currently not used
```

```
#pragma inline initUSBSlave
void initUSBSlave(void); // currently not used
```

After considering all the points in this section, the MCU hardware should be initialised and the low level firmware should work and can be called to further configure and use the hardware components of KNXcalibur.

5.2 Low level firmware

The software architecture should fulfil the following aspects:

- Usage of structured programming: All required functions to control the hardware shall be available by C-function calls.
- Portability: It should be possible to port the implemented software to other microcontrollers easily.
- Clear: The source code should be structured in such a way that the desired functionality can be located easily and a function fulfils a clear role.

To honor the constraints mentioned above, the firmware has been split into hardware dependent and hardware independent parts. A common header file can therefore have two corresponding C files. The hardware independent file has the same file name as the header file whereas the hardware dependent part has the same name with the controller family appended (e.g. `tpuart.h`, `tpuart.c`, `tpuart90330.c`). See Table 10 for a list of all files. The folder structure also has been created with this idea in mind and is the following:

```
fpraus/
   |---/common
          |---/h                          hardware independent header files
          |---/src                        hardware independent C files
   |---/MB90330
          |---/common
                 |---/h                   hardware dependent header files
                 |---/src                 hardware dependent C files
          |---/KNXcalibur
                 |---/common              common library project
                 |---/CS8900 Test         CS8900 test project
                 |---/CS8900 Test/Src     CS8900 test project files
                 |---/SD Test             SD/MMC card test project
                 |---/SD Test/Src         SD/MMC card test project files
                 |---/USB Test            USB test files
                 |---/TPUART Test         TP-UART test project
                 |---/TPUART Test/Src     TP-UART test project files
```

Header file	HW independent C file	HW dependent C file	Description
arp.h	arp.c	-	handle ARP requests and manage the ARP buffer
buffers.h	buffers.c	-	ringbuffers for Ethernet and ARP list
cemi.h	cemi.c	-	cEMI frame functions
config.h	-	config.c	configuration and globals for current microcontroller and setup
convert.h	convert.c	-	convert between EMI formats
cs8900.h	cs8900.c	cs890090330.c	functions to initialise and communicate with the CS8900A Ethernet controller
dhcp.h	dhcp.c	-	implementation of DHCP client
dos.h	-	dos90330.c	DOS-like in- and outport functions to communicate with devices at the emulated ISA bus
eib.h	eib.c	-	EIB-Frame functions
eibnetip.h	eibnetip.c	-	implementation of EIBnet/IP
eibserver.h	eibserver.c	-	EIBserver (BASys) functions
fat.h	fat.c	-	FAT16 implementation by Ulrich Radig
fat16.h	fat16.c	-	FAT16 implementation by Zoltan Gradwohl
globtype.h	-	-	global typedefs
helper.h	helper.c	helper90300.c	helper functions
http.h	http.c	-	implementation of an HTTP server
icmp.h	icmp.c	-	implementation of ICMP functions
ip.h	ip.c	-	implementation of IP protocol
mb90330.h	-	mb90330.asm	FFMC-16 I/O-Map
mmc.h	mmc.c	-	MMC/SD card functions
rtc.h	rtc.c	rtc90330.c	timer functions
slip.h	slip.c	slip90330.c	SLIP protocol
tcp.h	tcp.c	-	implementation of TCP protocol
tpuart.h	tpuart.c	tpuart90330.c	TP-UART driver
uartapi.h	uartapi.c	uart90330.c	UART functions
udp.h	udp.c	-	implementation of UDP protocol

Table 10: KNXcalibur: Overview of software files

5.2.1 Timer

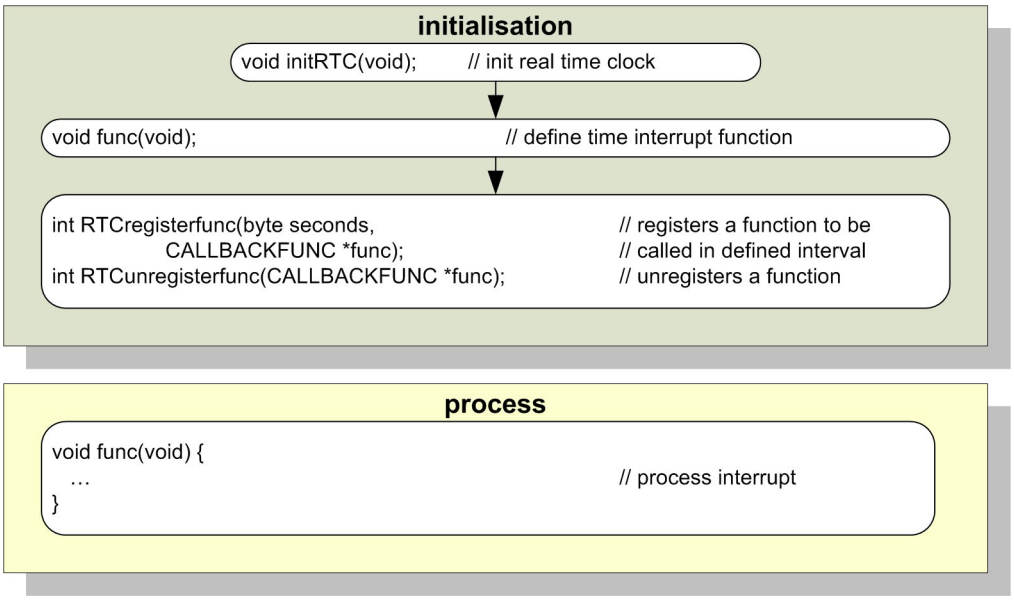


Figure 21: Timer callgraph

To provide easy and hardware independent access to the timers of the MCU, a dedicated timer function is implemented. It is primarily used to allow various functions (e.g. ARP timeout handling, EIBnet/IP timeout handling, …) to share a single timer with coarse resolution. The protocol implementations can register with a desired time interval and a callback function. Note that only non-blocking functions shall be registered, or otherwise the whole timer function will block. A hardware dependent function initialises a hardware timer to generate an interrupt every second. The ISR then iterates through the registered functions, checks the desired time interval of the callback and eventually calls the function.

5.2.2 UART

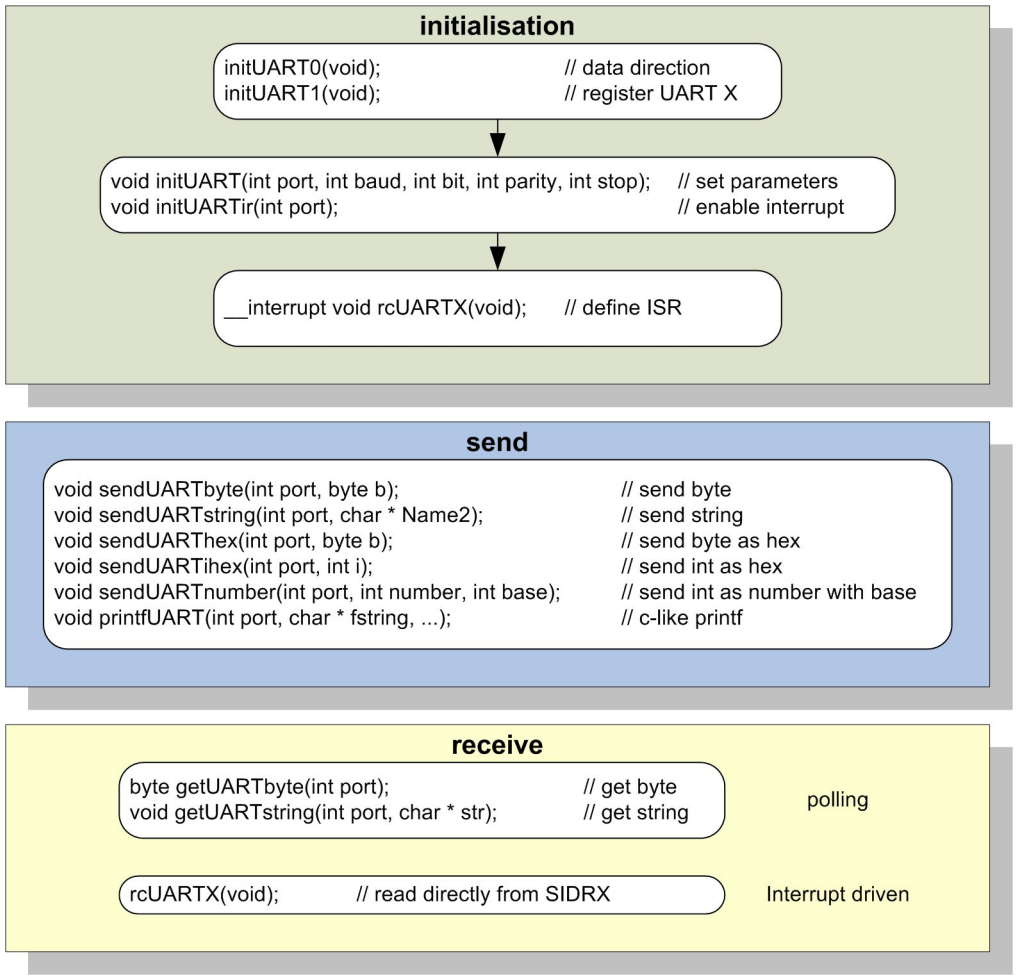


Figure 22: UART callgraph

The functions to access the serial interfaces form an important part of the firmware. On the one hand they are needed to communicate with the TP-UART chips and on the other hand they can be used for complex debugging or status messages with the help of a terminal program. During development without an emulator system they are often the only possibility to communicate with the MCU. The implemented firmware offers functions to initialise the interfaces with different parameters. Furthermore, functions allowing to output formatted text in

C-`printf` style exist. Communication can be interrupt driven or realised via polling. The implemented higher layers described in Section 5.3 only make use of interrupts. As seen in Table 10 hardware dependent and hardware independent parts exist. Only initialisation, sending and receiving of single bytes depend on the MCU used.

To provide TCP/IP capabilities without the use of an Ethernet controller, the SLIP protocol is implemented, which describes the transport of IP packets via a serial line. `UART1` is used to establish a dial-up connection with a SLIP host (e.g. Windows PC). All IP packets designated to KNXcalibur are received via `UART1` and stored in the same receiving buffer as those received from the CS8900A. For higher protocols it is irrelevant whether a packet is transmitted via the CS8900A or SLIP.

5.2.3 TP-UART

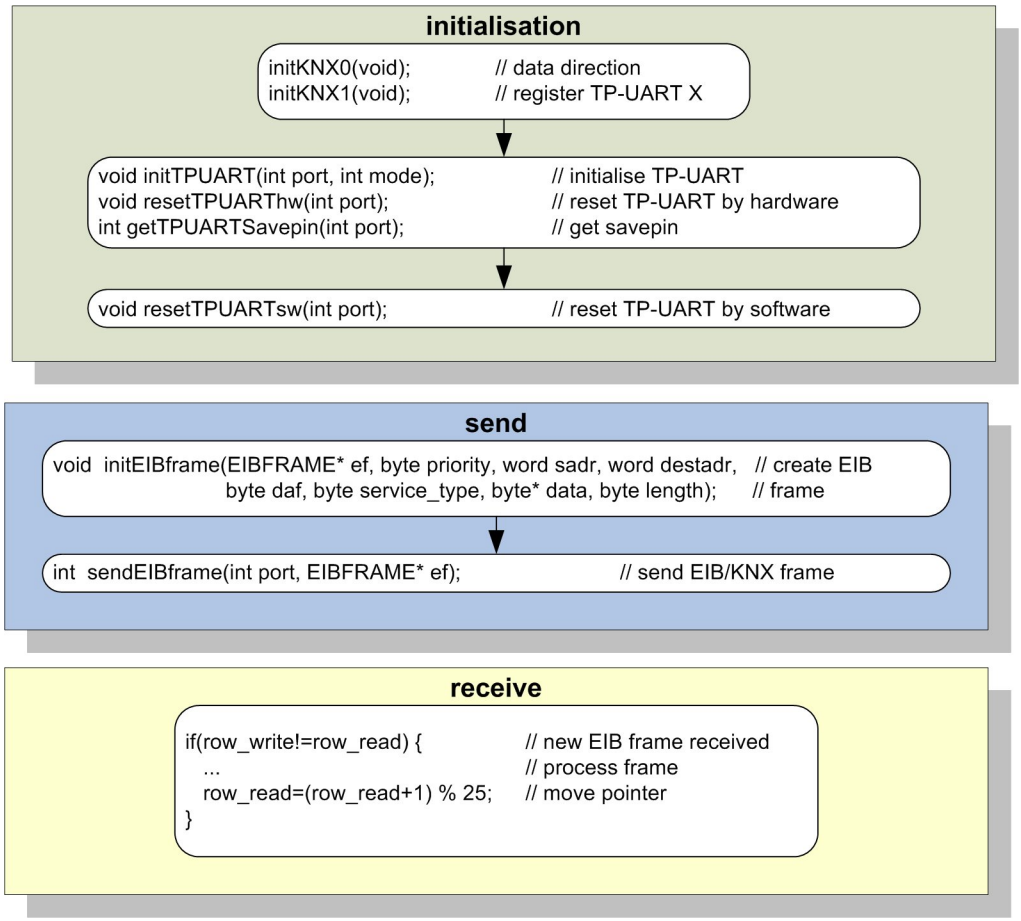


Figure 23: TP-UART callgraph

Two TP-UART ICs are present on KNXcalibur. It is possible to use one TP-UART in busmonitor mode for receiving and the second one for sending or both in normal mode for two different subnetworks. Moreover, `TPUART1` can be operated in analog mode.

For receiving KNX/EIB telegrams, a ring buffer for multiple telegrams with associated read and write pointers exist[23]: Buffer management is based on a two

[23] In the current implementation the ringbuffer stores up to 20 telegrams and is shared by both TP-UARTs.

dimensional field of bytes. Global variables indicate which slot in the ringbuffer is the one to write to and which is the one to read from. After a receive interrupt of a TP-UART, a service routine is called which stores the received data byte in the designated buffer. After that, a timer of the MCU is started in order to detect the end of a frame after 2.5 ms TP-UART silence. This timer is reset after every accepted byte. If it runs 2.5 ms – indicating the end of telegram – a different interrupt is generated, which marks the telegram as completely received and modifies the global pointers. Then a new telegram can be received in another position of the ring buffer and the previously received data can be processed.

Sending of an EIB frame is done without use of interrupts. The function `sendEIBframe()` implements the TP-UART handshake. Only one buffer exists, due to the fact that only one request can be processed at a time and hence multiple buffers would be useless.

In normal mode TP-UART implements a part of layer 2, simplifying KNX/EIB bus access. In analog mode, only the transceiver of the IC is working and the MB90F334A has immediate access to KNX/EIB bus level. This means that all timing constraints need to be considered and honored.

5.2.4 SD/MMC

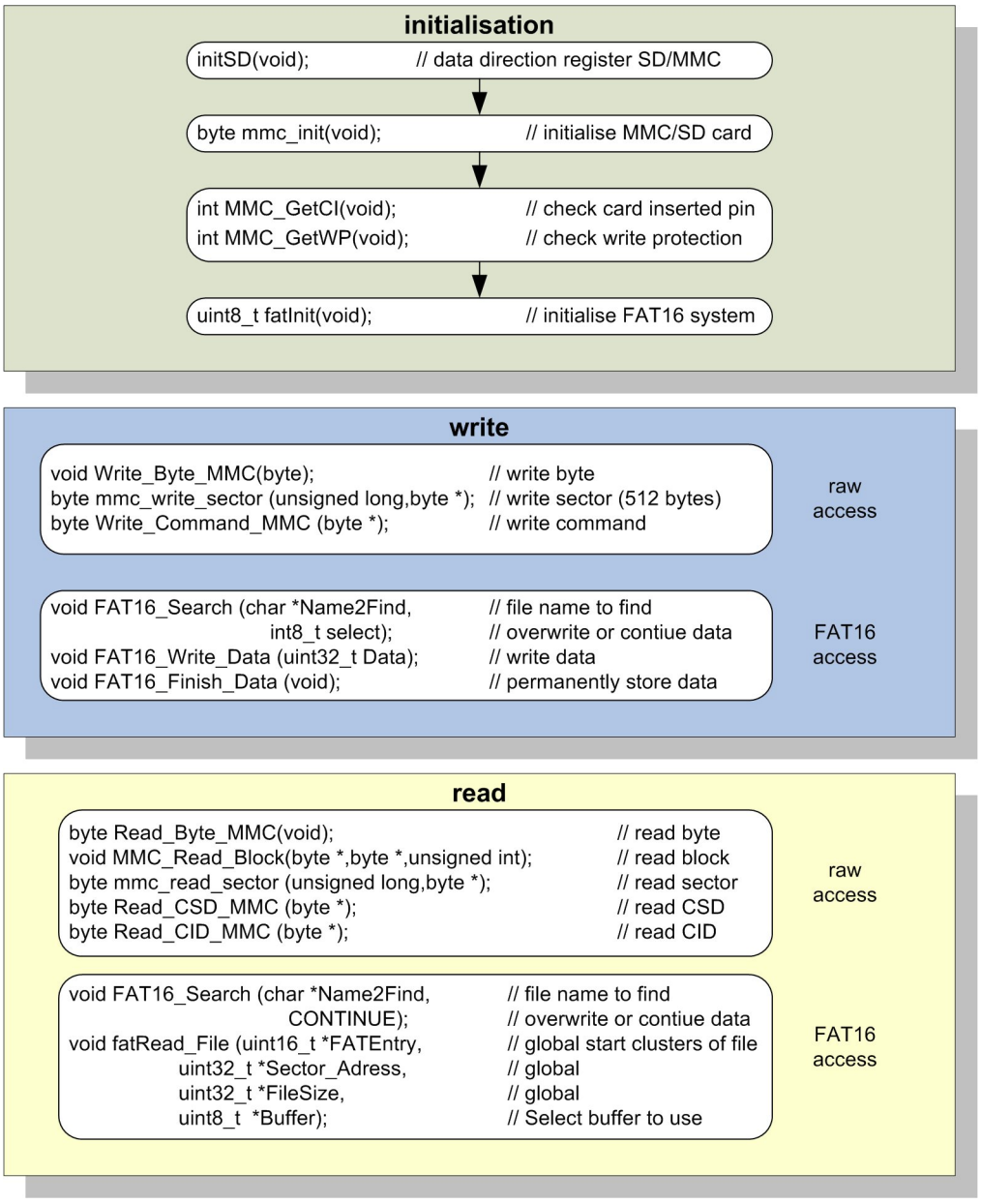


Figure 24: SD/MMC callgraph

SD/MMC access has been ported from Ulrich Radig and from Zoltan Gradwohl (see Appendix A.1 for Internet links). As mentioned earlier, the SD/MMC card is accessed via software SPI, meaning that all serial communication (e.g. outputting clock signal, determining logic level, ...) is handled in software. The timing constraints during, for instance, initialisation are met using busy waits. The speed of sending and receiving single bytes is only dependent on the speed of the MCU and the output of the clock signal. No wait states are present in the low-level communication routines.

The firmware supports all necessary parts of the SD/MMC protocol, starting with reading and writing bytes. Block and sector access is possible as well as reading the Card Identification Data (CID) and Card Specific Data (CSD) registers of the SD/MMC card.

On top of the low level functions, file systems like File Allocation Table (FAT)16, FAT32 or proprietary systems are possible. On KNXcalibur currently FAT16 read support and write support to existing files on the card is possible. This is quite sufficient for a simple web server or logging facilities. The SD/MMC card has to be prepared using, for example, Linux. With the help of `FDISK` a partition with file system (FAT16, type 06_h) is created. Afterwards the card has to be formatted using `MKDOSFS`.

5.2.5 CS8900A

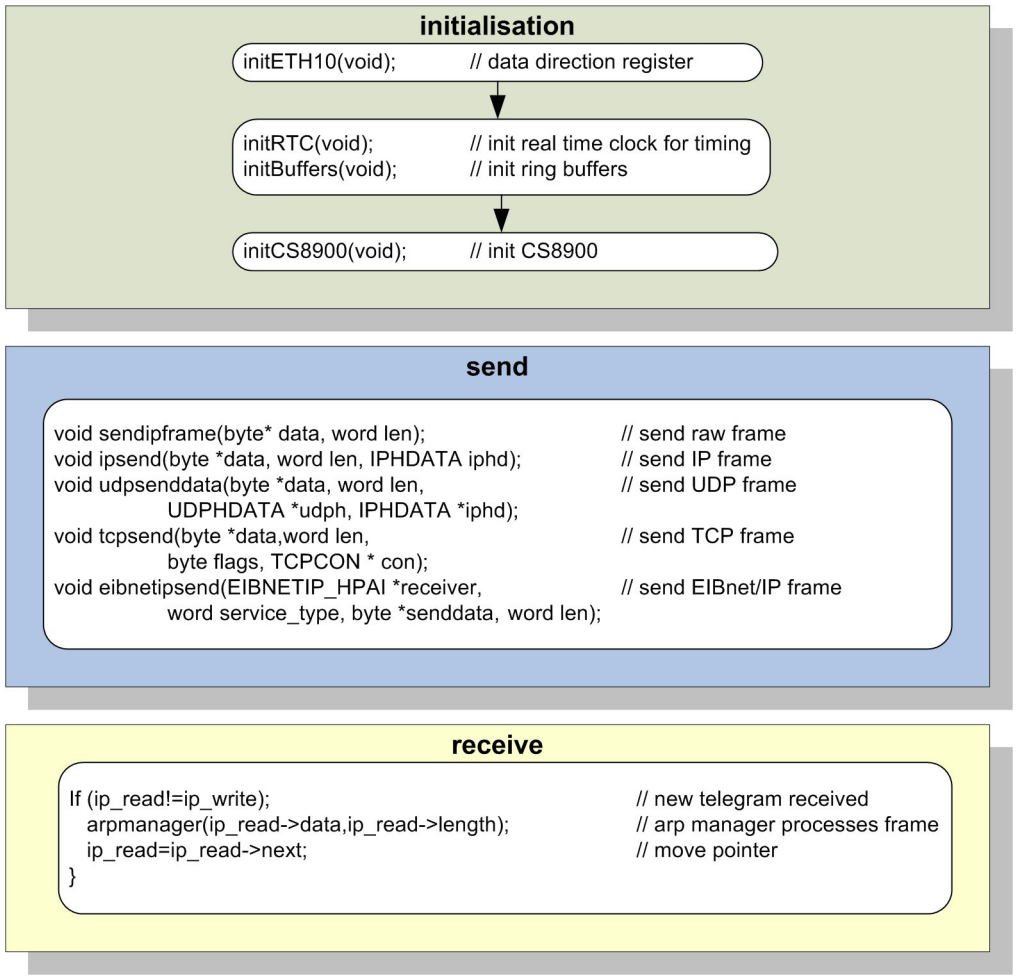


Figure 25: CS8900A callgraph

The Ethernet controller CS8900A is attached to the MCU via the described ISA emulation logic. To achieve high portability and compatibility with standard ISA bus programming methods, the well-known bus access functions `inport` and `outport` have been implemented. This additional encapsulation allows reuse of the initialisation and sending code on every CS8900A based network interface.

Receiving of Ethernet frames is interrupt driven – similar to the receiving of

KNX/EIB frames[24]. Therefore the interrupt out pin of the CS8900A is connected to the external interrupt pin of the MB90F334A. Currently interrupt pin 2 is used. This can be selected with other parameters in the file `config.h`. Transfer of a received Ethernet frame takes place in the ISR of the CS8900A, which is located in the file `cs890090330.c`. A ring buffer and the corresponding global pointers exist. For sending, only one send buffer is available (similar to the TP-UART).

The CS8900A controller is configured to pass packets to the MCU, which are sent to the broadcast address (FF:FF:FF:FF:FF) or to the address configured in its address register. This address is currently set to 00:00:08:15:47:11 and can also be configured in `config.h`. The ISR of the MCU usually should accept all received frames. It is, however, configured to pass only IP and ARP packets to higher layers in order to prevent the receiving buffer from overflowing and to minimise required processing power. This obviously breaches the protocol hierarchy, because not all packets are routed to their destination layer/protocol. Nevertheless this tradeoff has been accepted because a lot of unnecessary broadcasts (e.g. SAMBA communication) are filtered out.

5.2.6 USB

The Fujitsu USB firmware API (FUFA) and Fujitsu USB mini host API (FUMA) have been developed by Thesycon Systemsoftware & Consulting GmbH in cooperation with Fujitsu. Both libraries are free of charge and can be obtained from the webpage [71].

FUFA is a generic USB firmware library for microcontrollers of the MB90330 series implementing most of the functionality required by a USB function. It controls the USB function of the MB90330 and handles all standard USB requests. Requests like “suspend”, “resume” or “set configuration” are passed to the user application by means of callbacks. All device specific parameters like endpoint layout, protocol and USB descriptors can be controlled by the application software. All data transfer types (see Section 2.2.2) at standard and full speed are supported. Moreover, a demo application is included.

FUMA is a USB minihost library for Fujitsu MB90330 microcontrollers. It controls the USB minihost of the MCU and provides a programming interface (API) that is convenient to use. USB enumeration is completely covered by the library and USB events like “Device Attached” or “Device Removed” are passed on to the user application by means of call backs. A function-based interface to

[24]The ring buffer is able to store up to 5 frames

exchange data with a device is provided. No special device class is supported by the library itself. On top of it special classes like Mass Storage, HID or Printer class have to be implemented.

5.2.7 Test tools

To test a freshly manufactured and assembled board, several test tools have been developed. They offer simple test possibilities and make use of the low level firmware. These test tools have been assured to work and should be run on completed boards. The CS8900A, SD/MMC and TP-UART test programs offer a user interface via `UART1`. User input as well as output can be realised with a RS232 serial terminal (PC). No special UART test tool exists, since their correct function is a prerequisite (e.g. flashing, . . .) and can be observed using the other tools. The test tools are available as separate projects in the common workspace.

The *TP-UART test tool* allows to test both TP-UARTs in sending mode as well as in receiving mode. `TPUART1` can also be operated in analog mode. Since this mode has very though timing constraints, operating in analog mode is experimental. Using the test tool, the TP-UARTs can be reset by hardware. This means that an I/O pin of the MCU is connected to the external reset pin of the TP-UART. The Software then generates a high-low flank to reset the IC. A software reset is performed via the UART supported `TP_UART_Reset.req`. The status request corresponds to the TP-UART state. A logical ‘1’ is read from the associated pin if the TP-UART is is not connected to the KNX/EIB. A logical ‘0’ is read from the pin if it is connected to KNX/EIB. In the current implementation only one TP-UART can be in receive mode. Enabling receiving of one TP-UART automatically disables receiving of the other TP-UART. A packet sent to the same KNX/EIB line is received by the other TP-UART and output to the terminal. Figure 26 shows the user interface.

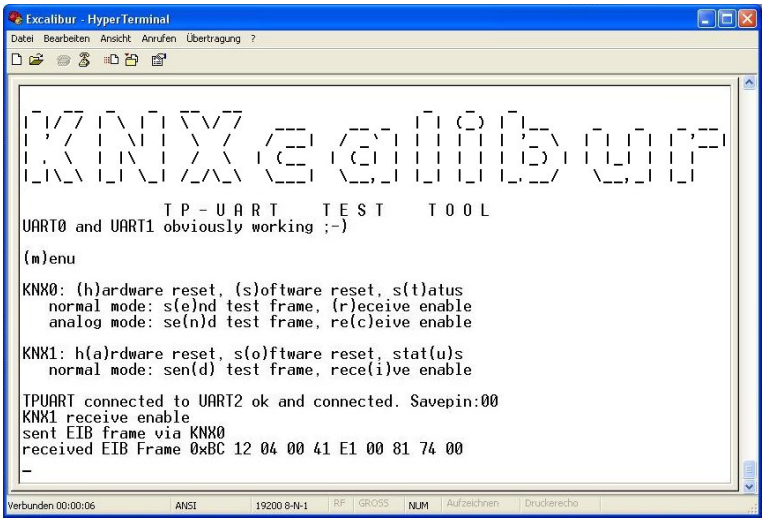


Figure 26: TP-UART test tool

The *SD/MMC test tool* performs functionality tests for the SD/MMC card and cardholder. The card has to be prepared as described in Section 5.2. The “card inserted” pin and the “write protected” pin allow to determine the state of the inserted card. If a card is inserted and is not write protected, a ‘0’ is read from both corresponding pins. Otherwise a ‘1’ is read. To access the SD/MMC card it has to be initialised at first. Then the CID and CSD (see [66]) as well as arbitrary blocks (512 bytes) can be read from the card. Moreover, a block at the arbitrarily chosen address 10 can be read and written by the tool. On top of this low level block access, the FAT16 file system can be tested: After initialising the FAT system, information about the partition and filesystem is printed. Then an `index.html` can be searched for and is printed to the terminal if found. Furthermore, some bytes can be appended to a file called `test.hex`. See Figure 27 for the user interface.

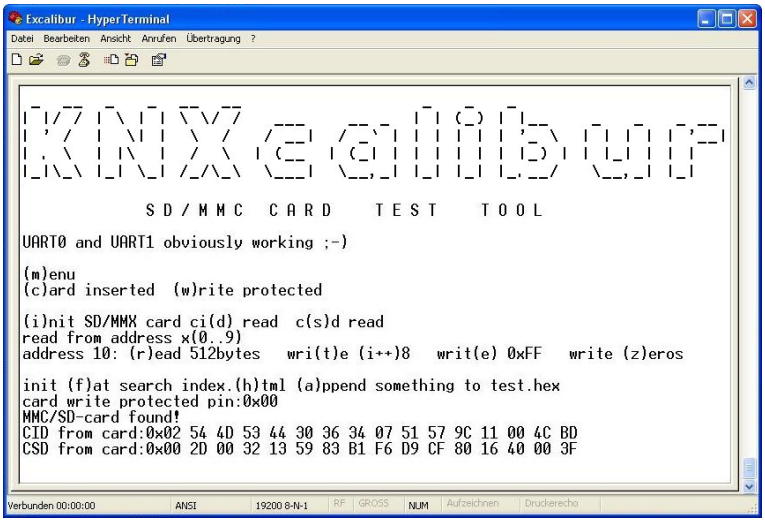


Figure 27: SD/MMC test tool

The *CS8900A test tool* is rather simple and allows no/minimal user interaction using the RS232 interface. It first tests communication with the Ethernet controller via a fixed test procedure (see [10]). The address lines as well as bus logic is tested in this step. If everything is fine (i.e. a fixed value can be read from a fixed address of the Ethernet controller), the test tool initialises the CS8900A. Then every received frame is passed to the higher stacks (see Section 5.3.1). KNXcalibur can now be accessed via the integrated web server or pinging (ICMP). The needed IP address is either obtained via DHCP or can be configured in and `config.h` `config.c`.

The firmware and demo applications of Thesycon have been selected as *USB test tools*. The required files are included in the USB test project and need to be unpacked, installed and compiled. To test the USB device function, the corresponding FUFA hex file has to be downloaded and the demo application has to be launched. After connecting KNXcalibur to the Windows PC, new hardware is found and the device driver is installed. Then simple strings can be transferred in an exchange with the demo application and a two way full speed transfer can take place. See Figure 28 for an overview.

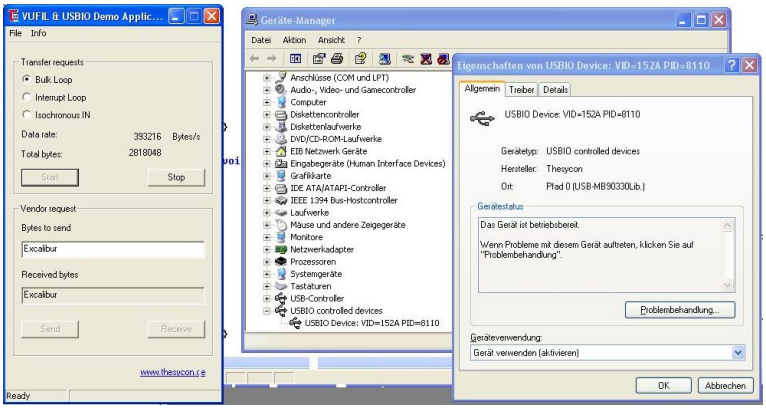


Figure 28: USB function test tool

To test the mini host function of KNXcalibur a USB keyboard is needed. After connecting an external power supply to the board, the USB keyboard can be plugged in. The FUMA mini host hex file needs to be compiled and transferred to the MCU. Then a serial connection with a terminal via `UART0` is established. KNXcalibur then simply echoes all pressed keys to the terminal.

5.3 Network protocol stacks

The firmware of the microcontroller is implemented in various stacks/layers and handlers. They make use of the low level firmware and extend the gateway's possibilities and protocol capabilities. The software is based on Oliver Alt's diploma thesis [4] and has been adapted, redesigned and improved. Regarding the IP stack, for instance, the firmware is now more user-friendly. Formerly, for sending each protocol layer had to be aware of the whole telegram length including length of all other protocols. Each layer had to calculate the position of its data in the sending buffer on its own. The IP layer, for instance, had to be aware of the size of the ARP header and write its data to position `sendbuffer[sizeof(ARPheader)]`. This handling has been improved using the principle explained in the next paragraph. Nevertheless, the firmware is far from being finished and a lot of work has to be invested into, for instance, securing the TCP/IP stack. The existing stacks, especially the EIBnet/IP layer interworking with ETS, form a proof of concept corroborating the design of KNXcalibur.

All layers feature the same mode of operation: The low level firmware transfers the data and stores it in a well known buffer (UART receive buffer, KNX/EIB

ringbuffer with associated read/write pointers, ...). This data is presented to the stacks as a byte array. By use of C language structures (typedef struct), pointers and casting, each network protocol stack can obtain the relevant data from the byte stream, without actually copying the data: The ARP handler, for example, defines the ARP header as structure (first 2 bytes=hardware type, second 2 bytes=protocol type, ...) and positions the corresponding pointer to the structure at the well known starting-point of a received frame (i.e. beginning of a frame received via the CS8900A) by use of a cast. It can then access data easily and eventually pass the next starting point for a different structure (e.g. IP) to the higher layer.

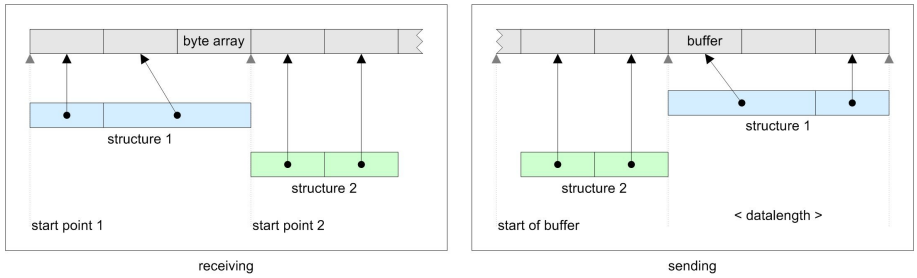


Figure 29: Network protocol stacks functioning principle

For sending frames one buffer for each physical network exists. Only one packet can be processed at a time and therefore a single buffer is sufficient. To minimise execution time and avoid usage of dynamic memory management the following approach has been chosen: The buffer is created statically with the size of the maximum transfer unit. The highest involved network protocol part starts filling the buffer from the end and passes the data length to the lower layer. This layer prepends its own data and again passes the request to the next layer. The physical layer then transmits the buffer to the network. The currently implemented network protocol stacks can be seen in Figure 30.

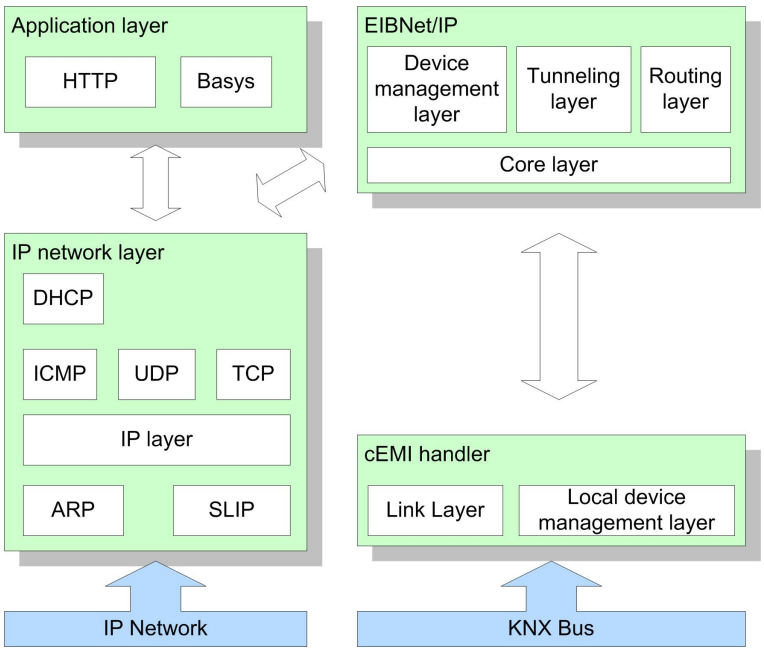


Figure 30: Software layer

5.3.1 IP

The IP stack forms the basis for many other network protocol stacks. [70] and [51] give a good introduction to this topic.

To connect the gateway to IP via the serial interface, the *SLIP protocol* is implemented. The gateway emulates a modem device using the standard protocol and so a connection with the host PC can be established: A new SLIP connection with arbitrary phone number but matching UART properties has to be created under dial up services in Windows 95/98 and Windows XP. The according connect function on the gateway then answers with “OK” to every received byte until the phone number is received. Then it responds with ”connect“, the connection is established and IP packets can be transferred via the send and receive function.

The other possibility to connect to the IP world is to use the CS8900A. When using a dedicated Ethernet controller, the *ARP* has to be implemented to map IP to Ethernet MAC addresses. The `arp.c` and `buffers.c` files contain the necessary code. The ARP-IP address mappings are stored in a ring buffer with FIFO principle – also called ARP list. Each mapping has a limited lifetime, which is managed in `buffers.c` by the timer described in Section 5.2.1. Upon receiv-

ing a packet via the Ethernet controller, the function `arpmanager` is called. If a MAC address as well as an IP address are present in a packet, the manager updates the ARP-IP mapping via a dedicated function in `buffers.c`. If the packets forms an `ARP_REQUEST` to the IP of the gateway, the manager processes the request and generates an `ARP_REPLY`. In any other case, the `arpmanager` passes the packet to the designated layer.

To send a frame via Ethernet, the function `sendipframe` is used. It queries the available ARP list for the ARP address, completes the packet and sends it via the CS8900A send function. It has to be mentioned that no `ARP_REQUEST` to the communication partner is sent if an ARP-IP mapping is not found in the list. Therefore every point-to-point communication must be initiated by the communication partner and not by the gateway. However, this is the case for all currently used network protocol stacks: To use BASys, EIBnet/IP (e.g. ETS) or a Hypertext Transfer Protocol (HTTP) client, they first have to establish a connection with the gateway so an ARP-IP mapping can be gained. EIBnet/IP routing, on the other hand, makes use of multicast communication and therefore the MAC address is set to FF:FF:FF:FF:FF. Hence this missing feature does not form a drawback or limitation.

The *IP* layer is located on top of the SLIP and ARP layer. Upon the reception of an IP package, it is handled by the function `ipmanager`. It checks the destination address and eventually passes the packet to the ICMP, UDP or TCP handler. Furthermore, it does checksum calculations. To send an IP frame the function `ipsend` is called, which completes the frame and calls the underlying network layer. IP fragmentation is currently not handled by the firmware.

ICMP was implemented mainly for testing purposes. The `icmpmanager` receives the packet from the IP layer. It checks the request and does checksum calculations. Currently only the `ICMP-ECHO-REQUEST - ICMP-ECHO-REPLY` handshake is implemented. The gateway responds to ICMP echo packets.

UDP is a stateless service and therefore easy to develop. It is fully implemented and is currently used for the BASys and EIBnet/IP part of the gateway. The `udpmanager` receives packets from the IP layer. Depending on the destination port, it calls the designated handlers, which are currently `eibrcvserver` and `eibnetipserver`. To send an UDP frame the function `udpsend` can be used.

For the moment, the *TCP* implementation is in a first and very experimental phase. The device only supports one connection to the destination port 80 (HTTP). This is sufficient for an experimental web server. The `tcpmanager` is responsible for establishing a TCP connection and eventually passing the requests

to higher layers like the HTTP layer. It does checksum calculation and discards faulty packets. To send a TCP frame the function `tcpsend` can be called.

For obtaining a dynamic IP address, *DHCP* is implemented. After powering up KNXcalibur, the firmware tries to obtain an IP address. If no DHCP server is present, the device uses a hard coded IP (192.168.1.50 in `config.c`).

5.3.2 Webserver

For the moment, HTTP (port 80) is the only implemented TCP service. Three applications are currently possible: The current device status (EIB connection, SD card status, ...) is returned in a formatted HTML page via the URL `/status` (e.g. http://192.168.1.50/status). Secondly a simple interface for sending KNX/EIB TP1 frames is presented via the URL `/send`. For any other URL requests a standard web server facility is implemented: The corresponding file is searched on the SD/MMC card and is then displayed to the user. It has to be mentioned that due to missing IP fragmentation the maximum file size is limited to the Maximum Transfer Unit (MTU) configured in `buffers.h`.

5.3.3 BASys integration

BASys makes use of a proprietary and simple protocol. It uses UDP on port 57776 as transport layer and offers three different services managed by `eibrcvserver`: Via the request 01_h the EIB gateway can be started. Every received frame from the KNX/EIB is then passed to the connected UDP client. Via the request 03_h the EIB gateway is stopped and via the request 10_h the attached frame is put to the EIB.

5.3.4 EIBnet/IP

Three EIBnet/IP implementations have been developed at the Automation Systems Group. Yet another EIBnet/IP gateway [60] is an implementation targeted at microcontroller use whereas Tweety [22] is a simple and modular implementation for PC-based devices. Eibd [42] is a daemon for Linux systems providing access to the KNX/EIB via PEI 10, PEI 16, TP-UART and EIBnet/IP. [60] is available since 2004, but has not been properly tested due to a missing EIBnet/IP client as communication partner. EIBnet/IP enabled ETS, more precisely the Falcon driver, had not been available at that time. Therefore the existing EIBnet/IP implementation was ported to KNXcalibur and tested. See Section 5.3.6 for a short overview

about the Tweety server and why it was not ported to KNXcalibur.

The EIBnet/IP stack is positioned on top of the IP network layer and on top of the cEMI handler and acts as a gateway between the two networks. It accepts EIBnet/IP frames from UDP/TCP level and cEMI frames from the cEMI handler. It processes the received frames according to their message types and passes valid packets to the destination service. The EIBnet/IP stack on this gateway consists of 4 fully implemented services, called layers in the following:

1. Core Layer: The EIBnet/IP Core Layer interprets the EIBnet/IP header and provides basic frame checks for incoming telegrams. If header size, message version and message size are correct, the telegram is passed to the specific handler. Furthermore, this layer is responsible for maintaining a list of currently active connections. Due to the fact that UDP, which does not provide any services for correct message ordering or message delivery, is used for transportation, the layer also has to handle these missing services. A sequence counter for incoming and outgoing messages is maintained by the layer. For timing driven services like heartbeats or timeouts, the timer firmware presented in Section 5.2.1 is used. The Core Layer supports multiple connections, which can be configured in `config.h`.

2. Device Management Layer: The EIBnet/IP Device Management layer is responsible for remote configuration and management of the device. It accepts EIBnet/IP packets carrying cEMI frames, checks them and passes the cEMI frames to the cEMI handler, which then processes the packets. The advantage of configuring the device over EIBnet/IP is the support of larger data structures.

3. Tunnelling Layer: EIBnet/IP Tunnelling is supported by the gateway in all three specified modes. Upon establishing a communication channel for tunnelling, the gateway assigns each tunnelling connection a KNX individual address. After establishing the communication channel, an EIBnet/IP client can send `TUNNELING_REQUEST` frames containing a KNX/EIB telegram. In Tunnelling on KNX Data Link Layer mode, only packets from the KNX bus with the destination address equal to the assigned address are passed to the EIBnet/IP client. Furthermore, all telegrams on KNX point-to-multipoint addressing are forwarded to the client. In cEMI Raw mode and KNX Busmonitor mode all received packets from the KNX network are passed to the EIBnet/IP client. Due to the fact that two KNX/EIB connections (i.e. two TP-UARTs) are present, a KNX busmonitor tunneling

connection can be maintained and at the same time the routing function is not affected.

4. Routing Layer: The gateway supports EIBnet/IP Routing. Filtering of telegrams according to their destination address is possible. KNXcalibur is able to handle messages from the IP network at 10 MBit/s (if the CS8900A is used). The KNX bus has a much slower speed and thus messages to KNX layer have to be queued. The messages are queued immediately after reception in a buffer. A single message is processed at a time. The buffersize is configurable in `buffers.h`. Currently the gateway features a queue of 20 frames to KNX layer and 5 frames to the IP network. The queue, however, can overflow resulting in message loss. The gateway then increments a counter and sends a routing lost message, which can be logged by a central supervising entity.

5.3.5 cEMI

KNXcalibur's internal EMI format is cEMI. Messages from the KNX/EIB are converted to this format to support easy integration into KNX on USB or EIBnet/IP.

The cEMI handler is responsible for handling incoming cEMI telegrams from the KNX bus and from the EIBnet/IP layer. Telegrams from the KNX side are saved in the receiving queue after an interrupt request from the TP-UART handler and are converted to cEMI before processing. Depending on the gateway state and the message type, the cEMI message is handled differently. If the gateway is in routing or tunnelling mode, packets with the corresponding destination addresses are passed from KNX/EIB Link Layer to the EIBnet/IP layer, which in turn generates EIBnet/IP frames and passes them to the IP layer. The IP layer then transmits the package to the IP network. If the EIBnet/IP handler is shutdown, only messages corresponding to the devices individual address are processed by the cEMI handler. In this case, the Link Layer is out of function and only Device Management via the KNX bus is possible.

The local Device Management layer interprets cEMI management frames, received by the KNX bus or by the EIBnet/IP layer. All gateway properties (compliant to [40] Part 3/8/3), like for example KNX individual address and device name can be configured. The layer is also responsible for generating the correct responses to the management server. For persistent storage, all data is kept in flash memory or on the SD/MMC card.

5.3.6 Tweety

Tweety [22] is a small, lean EIBnet/IP server and part of the KNXlive project. It is intended for people interested in their home automation system. Its architecture is well thought-out and Tweety has been tested to work with the ETS. So firstly the idea arose to port the software to KNXcalibur. But due to the following concerns, adapting the existing implementation [60] has been preferred:

- Tweety makes use of mechanisms like shared memory, pipes or semaphores. These techniques are not available on a microcontroller with limited RAM and CPU and missing OS. They would have to be emulated, which would not be very efficient.
- Only a single user is supported.
- Tweety only implements EIBnet/IP Tunnelling on link layer. No routing or Device Management services are possible. Hence the core and tunneling layers would have to be merged with the existing solution to support the full EIBnet/IP specification, which again would not be very efficient.

[60] therefore has been ported to KNXcalibur. It turned out that some bugs were present in the EIBnet/IP implementation. They have been fixed and now KNXcalibur supports the EIBnet/IP function of ETS.

6 Summary and outlook

The goal of this work was to produce a KNX/EIB gateway with versatile hardware as well as software interfaces. A compact solution based on a Fujitsu MB90330 microcontroller mounted on a eurocard format PCB has been designed. An extensible and cheap design with USB, Ethernet, RS232 and SD/MMC card support has been achieved. The ideas and proposals of [4] (miniaturisation, galvanic isolation of the KNX/EIB bus, implementation of DHCP, ...) have been integrated. The currently implemented low level firmware and network protocol stacks form a proof of concept for the hardware design.

Nevertheless, the gateway's firmware is far from being finished and further extensions to hard- and software are already in the author's mind:

- The existing stack design has to be improved. IP fragmentation needs to be implemented to be able to handle large data structures. Moreover, the web server should be able to server large files directly from the inserted SD/MMC card.

- The existing stacks, especially the IP stack, need to be secured. Currently buffer overflows due to manipulated packets are possible. Although limited possibilities exist to exploit these security flaws effectively, especially denial of service attacks based on buffer overflows form a severe threat.

- Connecting the inherently insecure KNX/EIB network to the real IP world is not recommended, as long as security questions like authentication and authorisation are unanswered. Basic work has been done in [32], but an existing secure solution does not exist yet. Therefore KNXcalibur should only be used in a trusted environment with underlying trusted communication buses. As an extension, cryptographic algorithms can be implemented and tested on KNXcalibur.

- KNX on USB is currently under development and will be integrated into the firmware of KNXcalibur.

- To support further extensions of the hardware, two pinheaders have been placed, which allow mounting of a daughter board in half-eurocard size on top of KNXcalibur. Extensions like additional LEDs and buttons or attaching an LCD are planned. Moreover, a PEI adapter circuit hardware can be placed on the daughter board.

- Currently the SPI pins of the MB90330 are used by the TP-UARTs and interrupt pins of the CS8900A. To use the hardware SPI of the MB90330, it should be possible to use jumpers for configuring the pin mapping. The reset pins of the TP-UARTs, for instance, should either be connected to the interrupt pins of the MB90330 or to normal I/Os.

List of Figures

List of Tables

Acronyms

A/D	Analog / Digital
ARP	Address Resolution Protocol
BACnet	Building Automation and Control Networking Protocol
BAS	Building Automation System
BAU	Bus Attachment Unit
BCU	Bus Coupling Unit
BIM	Bus Interface Module
BootP	Boot Protocol
CAN	Controller Area Network
CCMS	Centralised Control and Monitoring System
cEMI	common External Message Interface
CN	Control Network
CID	Card Identification Data
CPU	Central Processing Unit
CRC	Cyclical Redundancy Check
CRD	Connection Response Data Block
CRI	Connection Request Information
CSD	Card Specific Data
CSMA	Carrier Sense Multiple Access
DHCP	Dynamic Host Configuration Protocol
DIB	Description Information Block
DIP	Dual In-line Package

EEPROM	Electrically Erasable Programmable ROM
EMC	ElectroMagnetic Compatibility
EMI	External Message Interface
EMI	Electromagnetic Interference
ETS	EIB Tool Software
FAT	File Allocation Table
FUFA	Fujitsu USB firmware API
FUMA	Fujitsu USB mini host API
HAS	Home Automation System
HID	Human Interface Device
HPAI	Host Protocol Address Information
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air conditioning
I/O	Input / Output
I^2C	Inter-IC
IACK	Immediate Acknowledgement
IC	Integrated Circuit
ICMP	Internet Control Message Protocol
ICR	Interrupt Control Register
iETS	EIBlib/IP
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPR	Intellectual Property Rights

ISA	Industry Standard Architecture
ISR	Interrupt Service Routine
KNX	Konnex
KNX/EIB	European Installation Bus
LED	Light Emitting Diode
LON	LONWorks
LQFP	Low-profile Quad Flat Pack
MAC	Medium Access Control
MCU	Micro Controller Unit
MTU	Maximum Transfer Unit
OS	Operating System
OSGi	Open Services Gateway initiative
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PEI	Physical External Interface
PLC	Programmable Logic Controller
PPG	Pulse Pattern Generator
PWC	Pulse Width Count
RAM	Random Access Memory
RARP	Reverse Address Resolution Protocol
RCD	Residual Current Device
RF	Radio Frequency
RJ-45	Registered Jack - 45

ROM	Read Only Memory
SCADA	Supervisory Control And Data Acquisition
SD/MMC	Secure Digital / Multimedia Card
SLIP	Serial Line Internet Protocol
SMD	Surface Mounted Design
SOIC	Small-Outline Integrated Circuit
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TP-UART	Twisted Pair - Universal Asynchronous Receive Transmit
TP	Twisted Pair
TQFP	Thin Quad Flat Pack
UART	Universal Asynchronous Receive Transmit
UDP	User Datagram Protocol
USB	Universal Serial Bus
VCC	Voltage at the Common Collector
VPN	Virtual Private Network

References

[1] Accemic GmbH & Co. KG. [Online]. Available: http://www.accemic.com/

[2] C. E. Adams, "Home Area Network Technologies," *BT Technology Journal*, vol. 20, no. 2, pp. 53–72, 2002.

[3] *Single Channel, High Speed Optocouplers*, Agilent Technologies, 2001, Data Sheet. 5988-4111EN.

[4] O. Alt, "Entwurf und Realisierung einer EIB zu Ethernet Brücke (EIB-Gateway) in Hard- und Software," Tech. Rep., 2002.

[5] ——, "Entwicklung eines Softwaresystems zur Planung und Inbetriebnahme von Gebäudeautomationssystemen," Master's thesis, Technische Universität Darmstadt, 2003.

[6] *Control Network Protocol Specification*, ANSI/EIA/CEA Std. 709.1, Rev. A, 1999.

[7] *Control Network Protocol Specification*, ANSI/EIA/CEA Std. 709.1, Rev. B, 2002.

[8] *Tunneling Component Network Protocols Over Internet Protocol Channels*, ANSI/EIA/CEA Std. 852, 2002.

[9] *BACnet - A Data Communication Protocol for Building Automation and Control Networks*, ANSI/EIA/CEA Std. 135, 2004.

[10] D. Bodas, *Crystal LAN CS8900A Ethernet Controller Technical Reference Manual*, Cirrus Logic, 2001, Data Sheet.

[11] P. M. Bull, P. R. Benyon, and P. R. Limb, "Residential Gateways," *BT Technology Journal*, vol. 20, no. 2, pp. 73–81, 2002.

[12] S. T. Bushby, "BACnet: a standard communication infrastructure for intelligent buildings," in *Automation in Construction*, vol. 6, no. 5-6, 1997, pp. 529–540.

[13] CadSoft, *EAGLE Handbook 4.1*, 2004.

[14] CadSoft, *EAGLE Training Handbook 4.1*, 2004.

[15] B. Cain, S. E. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, “Internet Group Management Protocol, Version 3,” Internet Engineering Task Force, RFC 3376, Oct. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3376.txt

[16] *CS8900A Frequently Asked Questions*, Cirrus Logic, 2002, AN205.

[17] *Crystal LAN CS8900A Product Data Sheet*, Cirrus Logic, 2004, Data Sheet.

[18] Domoport. [Online]. Available: http://www.domoport.de/

[19] R. Droms, “Dynamic Host Configuration Protocol,” Internet Engineering Task Force, RFC 1531, Oct. 1993. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1531.txt

[20] *Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*, EIA Std. 232E, 1991.

[21] *Home and Building Electronic Systems (HBES)*, EN Std. 50 090, 1994-2004.

[22] B. Erb, G. Neugschwandtner, W. Kastner, and M. Kögler, “Open-source foundations for PC based KNX/EIB access and management,” in *Konnex Scientific Conference*. Technische Universität Wien, Institut für Automation, 2005.

[23] P. Fischer, “Analyse und Bewertung von Kommunikationssystemen in der Gebäudeautomation,” Ph.D. dissertation, Technische Universität Wien, Institut für Computertechnik, 2002.

[24] *F^2MC-16LX Family - Programming Flash MCUs*, Fujitsu Microelectronics Europe, 1999, Data Sheet. AN-FMEMCU-900031-22.

[25] *F^2MC-16LX Family - EMC Design Guide*, Fujitsu Microelectronics Europe, 2000, Data Sheet.

[26] *F^2MC-16LX Family - External Businterface*, Fujitsu Microelectronics Europe, 2000, Data Sheet. FMEMCU-AN-900034-19.

[27] *F^2MC-16LX Family - Hardware set up*, Fujitsu Microelectronics Europe, 2003, Data Sheet. FMEMCU-AN-900095-10.

[28] *F^2MC-16LX Family - Programming Flash MCUs 2*, Fujitsu Microelectronics Europe, 2003, Data Sheet. FMEMCU-AN-900095-10.

[29] *MB90330 series Data Sheet*, Fujitsu Microelectronics Europe, 2005, Data Sheet.

[30] *MB90330 series Hardware Manual*, Fujitsu Microelectronics Europe, 2005, Data Sheet.

[31] *EVBMB90F337 Manual*, GLYN GmbH & Co KG, Microcontroller Group, 2005, Data Sheet.

[32] W. Granzer, "Security in Networked Building Automation Systems," Master's thesis, Technische Universität Wien, Institut für Automation, 2005.

[33] *Telecontrol equipment and systems. Part 5: Transmission protocols*, IEC Std. 60 870-5, 1990.

[34] *Building Automation and Control Systems (BACS) - Part 5: Data Communication Protocol*, ISO Std. 16 484-5, 2003.

[35] D. L. Jones, *PCB Design Tutorial Rev. A*, 2004. [Online]. Available: http://www.alternatezone.com/

[36] W. Kastner and G. Neugschwandtner, "Service Interfaces for Field-Level Home and Building Automation," in *5th IEEE International Workshop on Factory Communication Systems*, September 2004, pp. 103–112.

[37] ——, "EIB: European Installation Bus," in *The Industrial Communication Technology Handbook*, ser. The Industrial Information Technology Series. CRC Press, 2005, vol. 1, ch. 34.

[38] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication Systems for Building Automation and Control," in *Proceedings of the IEEE*, vol. 93, no. 6, June 2005, pp. 1178–1203.

[39] A. Kethler and M. Neujahr, *Leiterplattendesign mit EAGLE*, 1st ed. mitp-Verlag, 2004.

[40] *KNX Handbook 1.1 and KNX Standard Extensions*, Konnex Association, Brussels, 2004.

[41] H. R. Kranz, *BACnet Gebäudeautomation 1.4*. Promotor, 2005.

[42] M. Kögler, "Free Development Environment for Bus Coupling Units of the European Installation Bus," Master's thesis, Technische Universität Wien, Institut für Automation, 2005.

[43] W. Lawrenz, *CAN - Controller Area Network*. Hüthig, 1997.

[44] D. Loy, D. Dietrich, and H. S. (Ed.), *Open Control Networks*. Kluwer Academic, 2001.

[45] *3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers Using Four 0.1μF External Capacitors*, Maxim Integrated Products, 1996, Data Sheet. 19-0273 Rev. 4.

[46] M. Mevenkamp and M. Mayer, "Energy efficiency in educational buildings using KNX/EIB," in *Konnex Scientific Conference*, 2005.

[47] *Single port up-right USB connector*, Molex Taiwan LTD., Data Sheet. SDA-89485-000.

[48] EIB Markt GmbH. [Online]. Available: http://www.eibmarkt.com/

[49] KNX/EIB Online Shop. [Online]. Available: http://www.knx-online-shop.de/

[50] Beta LAYOUT GmbH - PCB-POOL. [Online]. Available: http://www.pcbpool.de/

[51] D. Petrov, "WWW server in a chip," 2001. [Online]. Available: http://www.sxlist.com/techref/piclist/petrovwwwpic/index.htm

[52] *Hex inverter*, Philips Semiconductors, 1997, Data Sheet. 74LVC04A.

[53] *Quad 2-input OR gate*, Philips Semiconductors, 1997, Data Sheet. 74LVC32A.

[54] *Octal D-type transparent latch with 5-volt tolerant inputs/outputs (3-State)*, Philips Semiconductors, 1998, Data Sheet. 74LVC573A.

[55] D. C. Plummer, “Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48 bit Ethernet address for transmission on Ethernet hardware,” Internet Engineering Task Force, RFC 826, Nov. 1982. [Online]. Available: http://www.rfc-editor.org/rfc/rfc826.txt

[56] J. B. Postel, “User Datagram Protocol,” Internet Engineering Task Force, RFC 768, Aug. 1980. [Online]. Available: http://www.rfc-editor.org/rfc/rfc768.txt

[57] ——, “Internet Control Message Protocol,” Internet Engineering Task Force, RFC 792, Sept. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc792.txt

[58] ——, “Internet Protocol,” Internet Engineering Task Force, RFC 791, Sept. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt

[59] ——, “Transmission Control Protocol,” Internet Engineering Task Force, RFC 793, Sept. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[60] F. Praus, W. Kastner, and O. Alt, “Yet Another All-purpose EIBNet/IP Gateway,” in *Konnex Scientific Conference*. Technische Universität Wien, Institut für Automation, 2004.

[61] J. Romkey, “Nonstandard for transmission of IP datagrams over serial lines: SLIP,” RFC 1055 (Standard), Tech. Rep. 1055, June 1988. [Online]. Available: http://www.ietf.org/rfc/rfc1055.txt

[62] C. Sahm, *EIBlib/IP protocol Specification Version 1.1*, Konnex Association, Brussels, 2004.

[63] T. Saito, I. Tomoda, Y. Takabatake, K. Teramoto, and K. Fujimoto, “Gateway Technologies for Home Network and Their Implementations,” in *21st International Conference on Distributed Computing Systems*, 2001, pp. 175–180.

[64] T. Sauter, “The smart Fridge - A Networked Appliance,” in *Fieldbussystems and their Applications*, 2001, pp. 161–164.

[65] ——, “Integration Aspects in Automation - a Technology Survey,” in *10th IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 2, 2005, pp. 255–263.

[66] *MMC/SD - Description of HITACHI*, Data Sheet. [Online]. Available: http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/hitachi_hb28b128mm2.pdf

[67] *MMC/SD bus timing*, Data Sheet. [Online]. Available: http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/MMCSDTimming.pdf

[68] *EIB-TP-UART-IC*, Siemens, 10 2001, Data Sheet.

[69] *Very low drop voltage regulators with inhibit*, STMicroelectronics, 1998, Data Sheet. LF00AB/C.

[70] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[71] Thesycon Systemsoftware & Consulting GmbH. [Online]. Available: http://www.thesycon.de/

[72] C. Troger, "Realtime-Linux Device-Treiber für den europäischen Installationsbus," Master's thesis, Technische Universität Wien, 2002.

[73] *Single RJ45 connector module with integrated 10 base T magnetics and LEDs*, Umec elektronische Komponenten GmbH, 2002, Data Sheet. UE-LT1S023A-34.

[74] "USB Specification Version 2.0." [Online]. Available: http://www.usb.org/developers/docs.html

[75] "On-The-Go Supplement to the USB 2.0 Specification." [Online]. Available: http://www.usb.org/developers/onthego/

[76] A. Zaffran, "Physiktabellen: Kupferbreite - Strombelastbarkeit - Widerstand - Isolation - Temperaturabhängigkeit," 2003. [Online]. Available: ftp://ftp.cadsoft.de/pub/userfiles/doc/physik-tabellen-pcb.zip

A Appendix

A.1 Internet links

[KNXcalibur project homepage], *http://www.KNXcalibur.praus.at/*
[TU-Wien], *http://www.auto.tuwien.ac.at/knx/*

[Fujitsu microcontrollers], *http://www.fme.gsdc.de/gsdc.htm*
[Fujitsu Softune workbench], *http://www.fme.gsdc.de/products/softune0.htm*
[Thesycon Systemsoftware & Consulting GmbH], *http://www.thesycon.de/*

[AVR Ethernet], *http://www.ispf.de/modules.php?name=News&file=article&sid=5&page=0*
[Bascom-AVR], *http://members.home.nl/bzijlstra/software/examples/RTL8019as.htm*
[BASys], *http://www.basys2003.org/*
[Eagle 3D], *http://www.matwei.de/doku.php?id=en:eagle3d:eagle3d*
[EIB Userclub], *http://www.eib-userclub.de/*
[Ethernut], *http://www.ethernut.de*
[Gradwohl Zoltan], *http://www.mikrocontroller.net/forum/read-4-214168.html*
[Radig Ulrich], *http://www.ulrichradig.de/*

[ABB], *http://www.abb.de/*
[Adyna Technology GmbH], *http://www.adyna-tec.de/*
[Amann GmbH], *http://www.amann-net.de/*
[Aston GmbH], *http://www.aston-iport.de/*
[b.a.b-technologie gmbh], *http://www.bab-tec.de/*
[Daetwyler Cables+Systems], *http://www.daetwyler.net/*
[Disch GmbH], *http://disch-systems.de/*
[Domoport], *http://www.adyna-tec.de/*
[EIB Markt GmbH], *http://www.eibmarkt.com/*
[ELKA Elektronik GmbH], *http://www.elka.de/*
[ESF Software GmbH], *http://www.esf-software.com/*
[Gira homeserver], *http://dacom-homeautomation.de/*
[Hager], *http://www.hager.de/*
[IT–Gesellschaft für Informationstechnik], *http://www.it-gmbh.de/*
[Albrecht Jung GmbH & Co. KG], *http://www.jung.de/*
[KNX/EIB Online Shop], *http://www.knx-online-shop.de/*
[NETxAutomation Software GmbH], *http://netxautomation.com/*
[Schlaps&Partner], *http://www.schlaps-automation.de/*

A.2 MB90330: Pin description

Pin No.	Pin Name	Circuit Type	Functional description
107	X1	A	It is oscillation terminal.
108	X0	A	It is oscillation terminal.
13	X0A	A	It is 32 kHz oscillation terminal.
14	X1A	A	It is 32 kHz oscillation terminal.
90	$\overline{\text{RST}}$	F	It is reset input.
93 to 100	P00 to P07	H	It is General-purpose I/O port. You can set a pull-up resistor ON (RD00 to RD07= "1") with the pull-up resistor setting register (RDR0) (When the power output is set, it is invalid).
	AD00 to AD07		Functions as an I/O pin for the low-order external address/data bus in multiplex mode.
	D00 to D07		Functions as an output pin for the low-order external data bus in non-multiplex mode.
101 to 104	P10 to P13	H	It is General-purpose I/O port. You can set a pull-up resistor ON (RD10 to RD13= "1") with the pull-up resistor setting register (RDR1) (When the power output is set, it is invalid).
	AD08 to AD11		Functions as an I/O pin for the high-order external address/data bus in multiplex mode.
	D08 to D11		Functions as an output pin for the high-order external data bus in non-multiplex mode.
109 to 112	P14 to P17	H	It is General-purpose I/O port. You can set a pull-up resistor ON (RD14 to RD17= "1") with the pull-up resistor setting register (RDR1) (When the power output is set, it is invalid)
	AD12 to AD15		Functions as an I/O pin for the high-order external address/data bus in multiplex mode.
	D12 to D15		Functions as an output pin for the high-order external data bus in non-multiplex mode.

Figure 31: MB90330: Pin function 1/7 (from [30])

Pin No.	Pin Name	Circuit Type	Functional description
113 to 116	P20 to P23	D	It is General-purpose input/output port. Functions as the general-purpose input/output port in the external bus mode if the bit corresponding to external address output control register (HACR) is set to "1".
	A16 to A19		Functions as the upper output pin of an address (A16 to A19) in the multiplex mode if the bit corresponding to external address output control register (HACR) is set to "0".
	A16 to A19		Functions as the upper output pin of an address (A16 to A19) in the non-multiplex mode if the bit corresponding to external address output control register (HACR) is set to "0".
117 to 120	P24 to P27	D	It is General-purpose input/output port. Functions as the general-purpose input/output port in the external bus mode if the bit corresponding to external address output control register (HACR) is set to "1".
	A20 to A23		Functions as the upper output pin of an address (A20 to A23) in the multiplex mode if the bit corresponding to external address output control register (HACR) is set to "0".
	A20 to A23		Functions as the upper output pin of an address (A20 to A23) in the non-multiplex mode if the bit corresponding to external address output control register (HACR) is set to "0".
	PPG0 to PPG3		Function as ch0 to ch3 output pins for the PPG timer.
1	P30	D	It is General-purpose I/O port.
	A00		Functions as the external address pin in non-multi-bus mode.
	TIN1		Functions as an event input pin for 16-bit reload timer ch1.
2	P31	D	It is General-purpose I/O port.
	A01		Functions as the external address pin in non-multi-bus mode.
	TOT1		Functions as an output pin for 16-bit reload timer ch1.
3	P32	D	It is General-purpose I/O port.
	A02		Functions as the external address pin in non-multi-bus mode.
	TIN2		Functions as an event input pin for 16-bit reload timer ch2.
4	P33	D	It is General-purpose I/O port.
	A03		Functions as the external address pin in non-multi-bus mode.
	TOT2		Functions as an output pin for 16-bit reload timer ch2.
5 to 8	P34 to P37	D	It is General-purpose I/O port.
	A04 to A07		Functions as the external address pin in non-multi-bus mode.

Figure 32: MB90330: Pin function 2/7 (from [30])

Pin No.	Pin Name	Circuit Type	Functional description
9	P40	G	It is General-purpose I/O port.
	A08		Functions as the external address pin in non-multi-bus mode.
	TIN0		Functions as an event input pin for 16-bit reload timer ch0.
10	P41	G	It is General-purpose I/O port.
	A09		Functions as the external address pin in non-multi-bus mode.
	TOT0		Functions as an output pin for 16-bit reload timer ch0.
11	P42	G	It is General-purpose I/O port.
	A10		Functions as the external address pin in non-multi-bus mode.
	SIN0		Functions as a data input pin for UART ch0.
12	P43	G	It is General-purpose I/O port.
	A11		Functions as the external address pin in non-multi-bus mode.
	SOT0		Functions as a data output pin for UART ch0.
17	P44	G	It is General-purpose I/O port.
	A12		Functions as the external address pin in non-multi-bus mode.
	SCK0		Functions as a clock I/O pin for UART ch0.
18	P45	G	It is General-purpose I/O port.
	A13		Functions as the external address pin in non-multi-bus mode.
	SIN1		Functions as a data input pin for UART ch1.
19	P46	G	It is General-purpose I/O port.
	A14		Functions as the external address pin in non-multi-bus mode.
	SOT1		Functions as a data output pin for UART ch1.
20	P47	G	It is General-purpose I/O port.
	A15		Functions as the external address pin in non-multi-bus mode.
	SCK1		Functions as a clock I/O pin for UART ch1.
81	P50	L	It is General-purpose I/O port.
	ALE		Functions as the address latch enable signal (ALE) pin in external bus mode.
82	P51	L	It is General-purpose I/O port.
	$\overline{RD}$		Functions as the read strobe output (RDX) pin in external bus mode.

Figure 33: MB90330: Pin function 3/7 (from [30])

Pin No.	Pin Name	Circuit Type	Functional description
83	P52	L	It is General-purpose I/O port.
	$\overline{\text{WRL}}$		Functions as the data write strobe output (WRLX) pin on the lower side in external bus mode. Functions as a general-purpose I/O port when the WRE bit in the EPCR register is "0".
84	P53	L	It is General-purpose I/O port.
	$\overline{\text{WRH}}$		Functions as the data write strobe output (WRHX) pin on the higher side in 16-bit external bus mode. Functions as a general-purpose I/O port when the WRE bit in the EPCR register is "0".
85	P54	L	It is General-purpose I/O port.
	HRQ		Functions as the hold request input (HRQ) pin in external bus mode. Functions as a general-purpose I/O port when the HDE bit in the EPCR register is "0".
86	P55	L	It is General-purpose I/O port.
	$\overline{\text{HAK}}$		Functions as the hold acknowledge output (HAKX) pin in external bus mode. Functions as a general-purpose I/O port when the HDE bit in the EPCR register is "0".
91	P56	L	It is General-purpose I/O port.
	RDY		Functions as the external ready input (RDY) pin in external bus mode. Functions as a general-purpose I/O port when the RYE bit in the EPCR register is "0".
92	P57	L	It is General-purpose I/O port.
	CLK		Functions as the machine cycle clock output (CLK) pin in external bus mode. Functions as a general-purpose I/O port when the CKE bit in the EPCR register is "0".
21 to 22	P60 to P61	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT0 to INT1		Function as input pins for external interrupt ch0 to ch1.
23	P62	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT2		Function as input pins for external interrupt ch2.
	SIN		It is simple serial I/O data output terminal.
24	P63	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT3		Function as input pins for external interrupt ch3.
	SOT		It is simple serial I/O data output terminal.
25	P64	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT4		Function as input pins for external interrupt ch4.
	SCK		It is simple serial I/O clock input/output terminal.

Figure 34: MB90330: Pin function 4/7 (from [30])

Pin No.	Pin Name	Circuit Type	Functional description
26	P65	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT5		Function as input pins for external interrupt ch5.
	PWC		Functions as the PWC input pin.
27	P66	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT6		Function as input pins for external interrupt ch6.
	SCL0		Functions as the clock I/O pin for the I^2C interface ch0. Set port output to Hi-Z during I^2C interface operations.
28	P67	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	INT7		Function as input pins for external interrupt ch7.
	SDA0		Functions as the data I/O pin for the I^2C interface ch0. Set port output to Hi-Z during I^2C interface operations.
39 to 46	P70 to P77	I	It is General-purpose I/O port.
	AN0 to AN7		Function as input pins for analog ch0 to ch7.
48 to 55	P80 to P87	I	It is General-purpose I/O port.
	AN8 to AN15		Function as input pins for analog ch8 to ch15.
29	P90	D	It is General-purpose I/O port.
	SIN2		Functions as a data input pin for UART ch2.
30	P91	D	It is General-purpose I/O port.
	SOT2		Functions as a data output pin for UART ch2.
31	P92	D	It is General-purpose I/O port.
	SCK2		Functions as a clock I/O pin for UART ch2.
32	P93	D	It is General-purpose I/O port.
	SIN3		Functions as a data input pin for UART ch3.
33	P94	D	It is General-purpose I/O port.
	SOT3		Functions as a data output pin for UART ch3.
34	P95	D	It is General-purpose I/O port.
	SCK3		Functions as a clock I/O pin for UART ch3.
35	P96	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	ADTG		Functions as the external trigger input pin when the A/D converter is being used.
	FRCK		Functions as the external clock input pin when the free-run timer is being used.

Figure 35: MB90330: Pin function 5/7 (from [30])

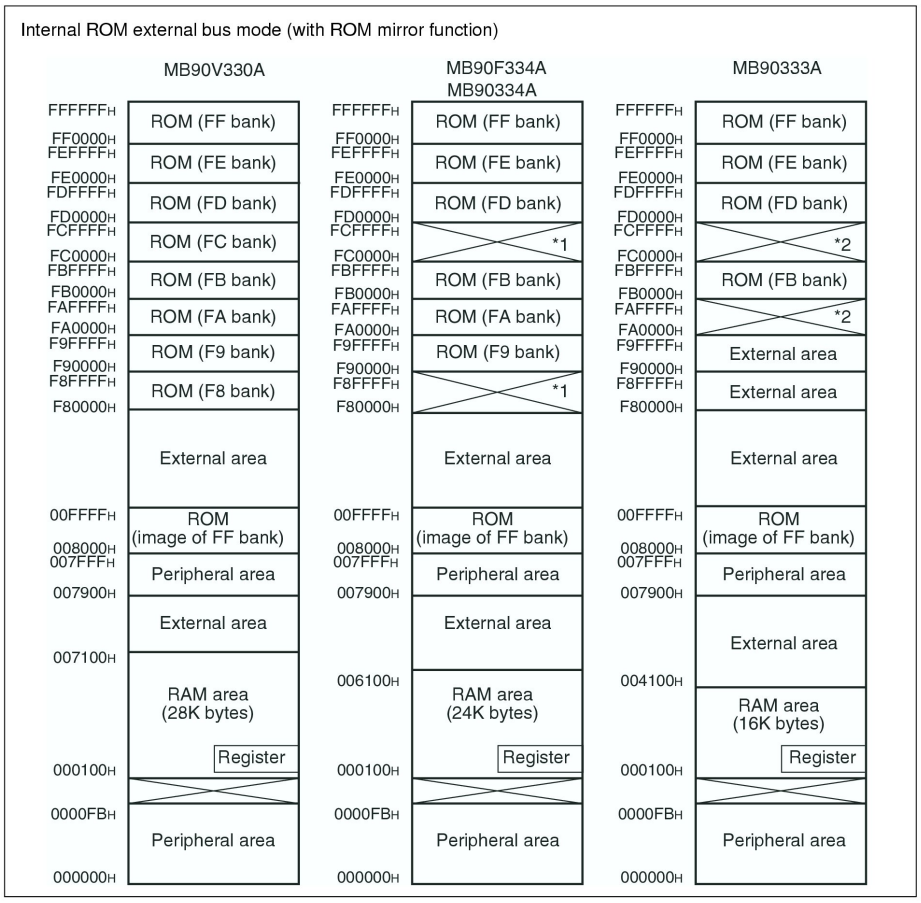
Pin No.	Pin Name	Circuit Type	Functional description
56 to 59	PA0 to PA3	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	IN0 to IN3		Captures as trigger input for ch0 to ch3 of the input capture.
60 to 63	PA4 to PA7	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	OUT0 to OUT3		Functions as the event output pins for ch0 to ch3 of the output compare.
64	PB0	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	SCL1		Functions as the clock I/O pin for the I^2C interface ch1. Set port output to Hi-Z during I^2C interface operations.
65	PB1	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	SDA1		Functions as the data I/O pin for the I^2C interface ch1. Set port output to Hi-Z during I^2C interface operations.
66	PB2	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	SCL2		Functions as the clock I/O pin for the I^2C interface ch2. Set port output to Hi-Z during I^2C interface operations.
67	PB3	C	It is General-purpose I/O port (Withstand voltage of 5 V).
	SDA2		Functions as the data I/O pin for the I^2C interface ch2. Set port output to Hi-Z during I^2C interface operations.
68	PB4	C	It is General-purpose I/O port (Withstand voltage of 5 V).
69	PB5	D	It is General-purpose I/O port.
	PPG4		Function as ch4 output pins for the PPG timer.
70	PB6	D	It is General-purpose I/O port.
	PPG5		Function as ch5 output pins for the PPG timer.
71	VBUS	C	Terminal for state detection of USB cable (Withstand voltage of 5 V).
73	DVM	K	It is USB Function D- terminal.
74	DVP	K	It is USB Function D + terminal.
77	HVM	K	It is USB Mini-HOST D- terminal.
78	HVP	K	It is USB Mini-HOST D + terminal.
80	$\overline{\text{HCON}}$	E	It is external pull-up resistor pin.
36	AVcc	-	It is A/D converter power supply pin.
37	AVRH	J	It is A/D converter external reference power supply pin.
38	AVss	-	It is A/D converter power supply pin.
87 to 89	MD2 to MD0	B	It is Operation mode select input pin.

Figure 36: MB90330: Pin function 6/7 (from [30])

Pin No.	Pin Name	Circuit Type	Functional description
15	Vcc	-	It is power supply pin.
75	Vcc	-	It is power supply pin.
79	Vcc	-	It is power supply pin.
105	Vcc	-	It is power supply pin.
16	Vss	-	It is power supply pin (GND).
47	Vss	-	It is power supply pin (GND).
72	Vss	-	It is power supply pin (GND).
76	Vss	-	It is power supply pin (GND).
106	Vss	-	It is power supply pin (GND).

Figure 37: MB90330: Pin function 7/7 (from [30])

A.3 MB90330: Memory map



*1: In the area of $F80000_H$ to $F8FFFF_H$ and $FC0000_H$ to $FCFFFF_H$ at MB90F334, MB90334A a value of "1" is read at read operating.

*2: In the area of $FA0000_H$ to $FAFFFF_H$ and $FC0000_H$ to $FCFFFF_H$ at MB90333, a value of "1" is read at read operating.

Figure 38: MB90330: Memory map (from [30])

A.4 KNXcalibur: Schematic diagram

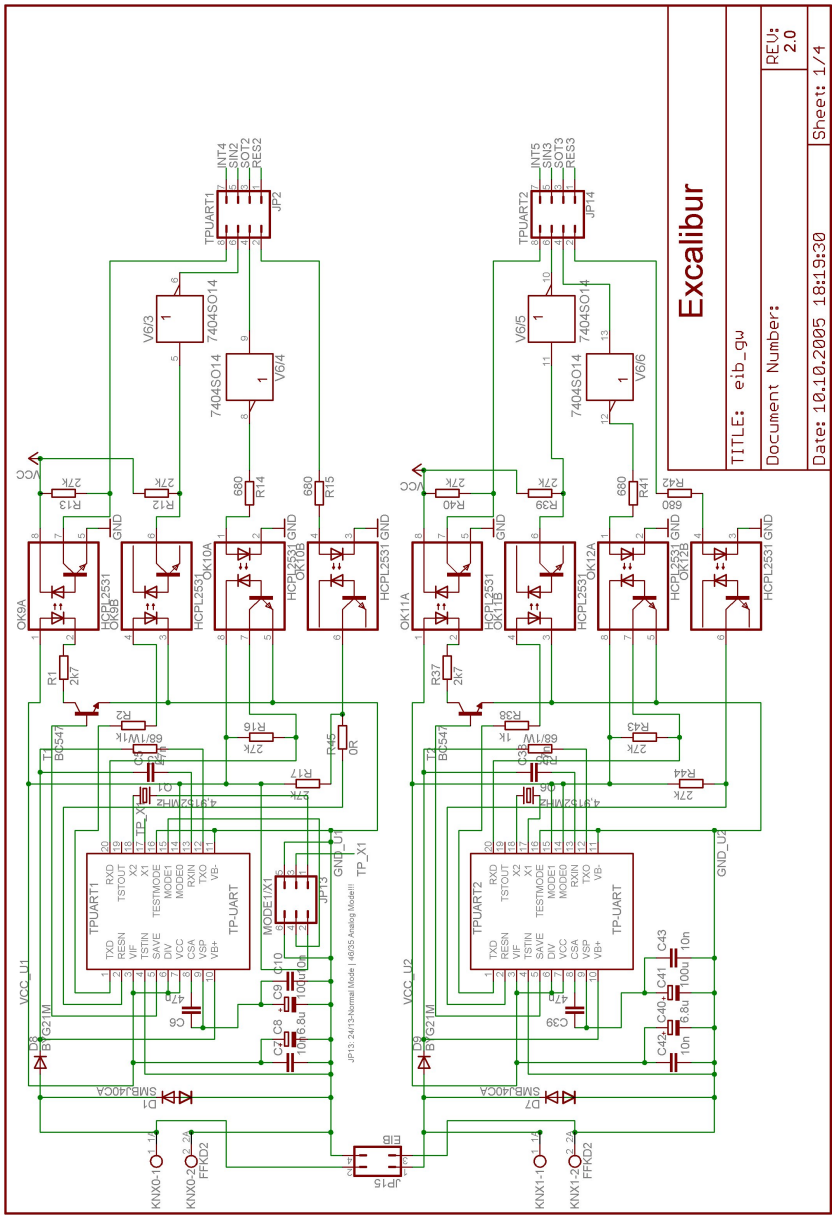


Figure 39: KNXcalibur: Schematic 1/4

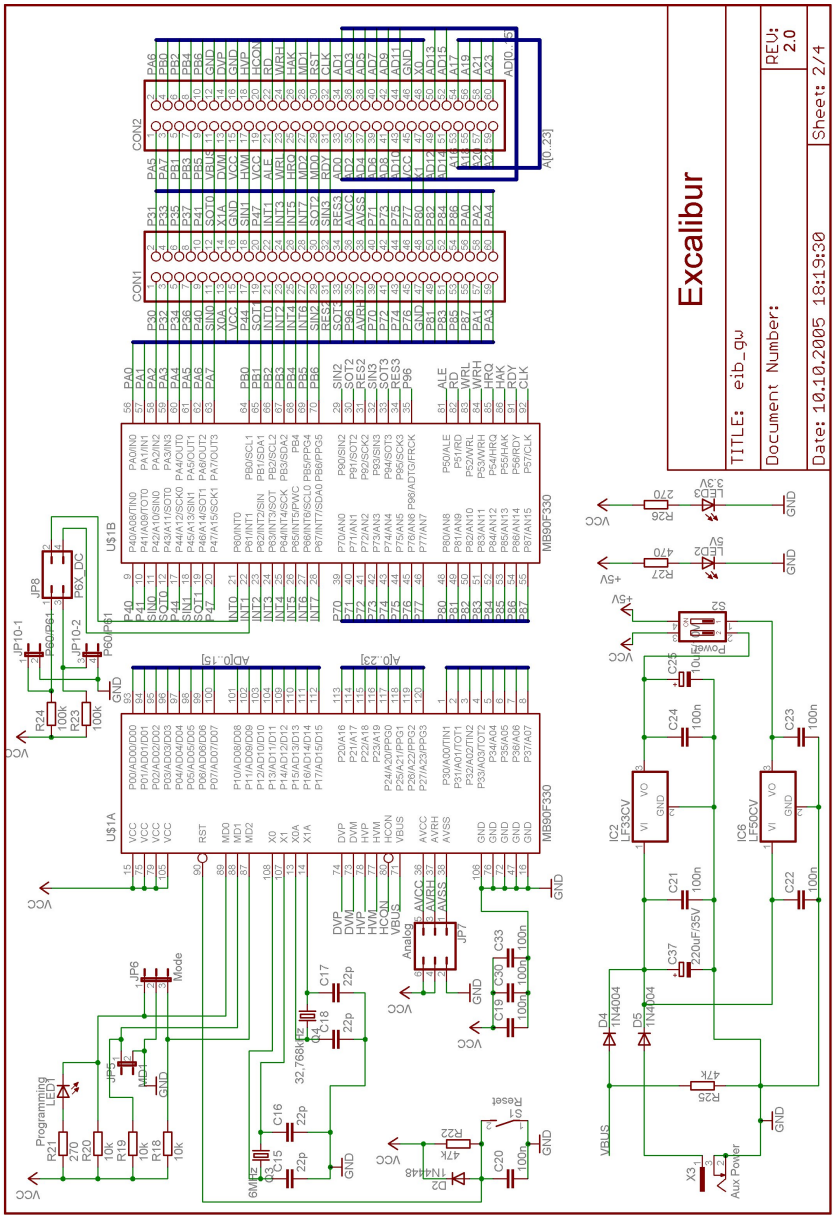


Figure 40: KNXcalibur: Schematic 2/4

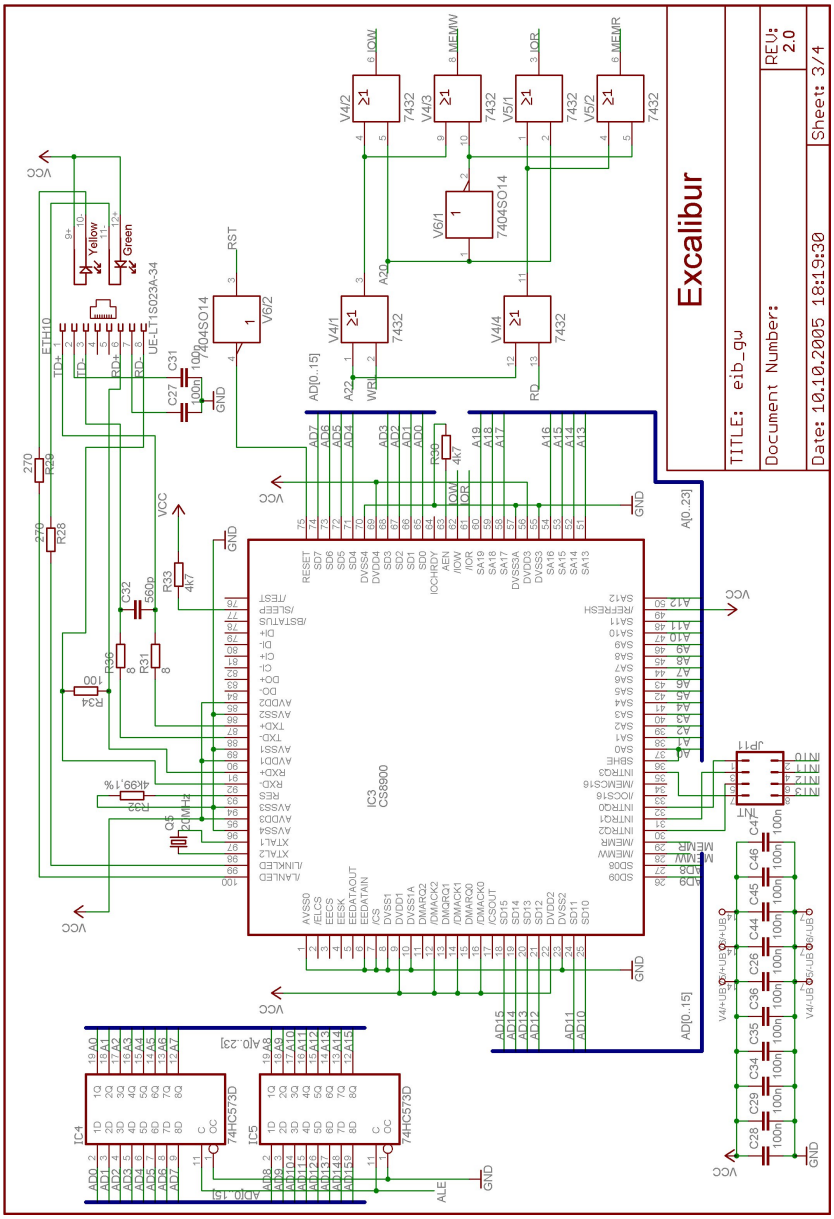


Figure 41: KNXcalibur: Schematic 3/4

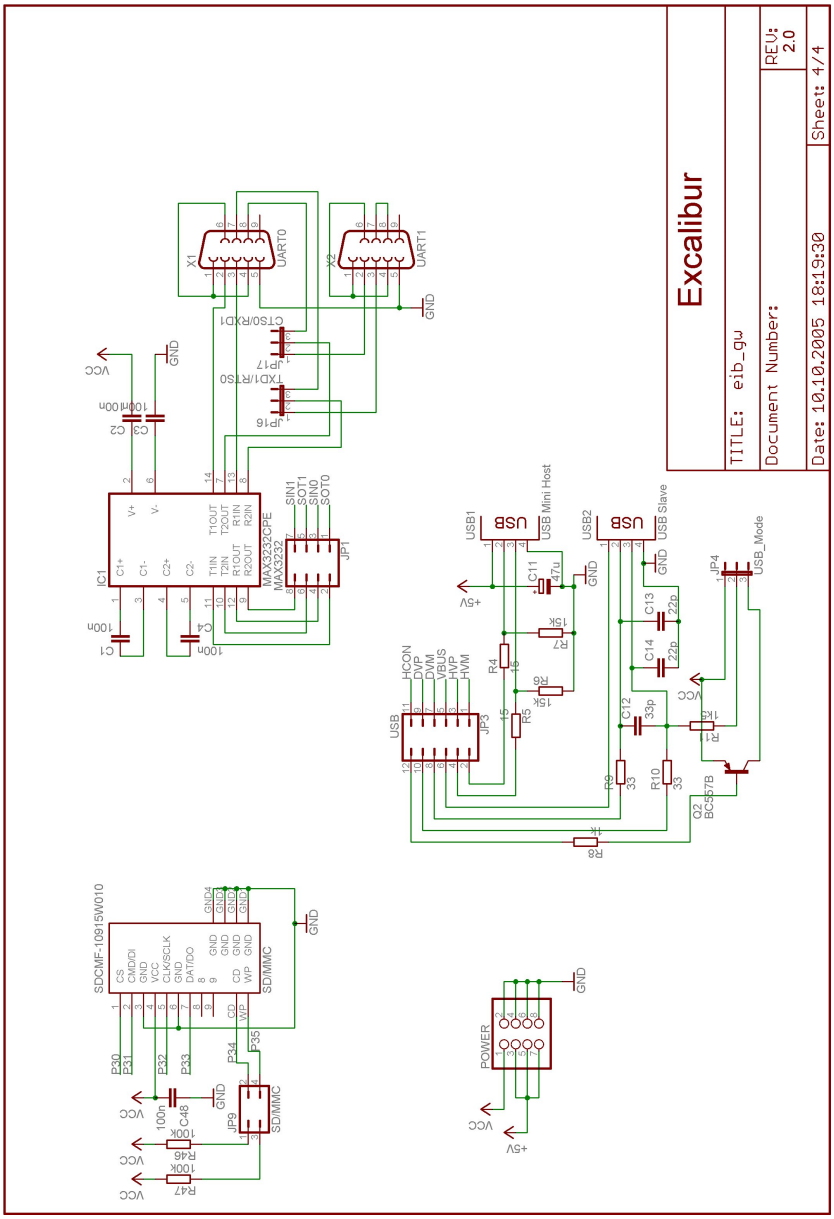


Figure 42: KNXcalibur: Schematic 4/4

A.5 KNXcalibur: Part list

Amount	Value	Device	Component	RS/Farnell-Item nr.
1		PINHD-2X4	POWER	251-8648
2		PINHD-2X30	CON1, CON2	251-8648
1	0R	R-EU_R1206	R45	351-3199
2	1N4004	1N4004	D4, D5	261-176
1	1N4448	1N4446	D2	885101
3	1k	R-EU_R1206	R2, R8, R38	223-2265
1	1k5	R-EU_R1206	R11	223-2287
2	2k7	R-EU_R1206	R1, R37	223-2322
1	3.3V	LED3MM	LED3	228-4979
2	4,9152MHz	CRYTALHC49U-V	Q1, Q6	657-577
2	4k7	R-EU_R1206	R30, R33	223-2350
1	4k99,1%	R-EU_R0805	R32	215-3162
1	5V	LED3MM	LED2	228-4979
2	6.8μ	CPOL-EU153CLV-0405	C8, C40	197-8386
1	6MHz	CRYTALHC49U-V	Q3	657-583
2	8	R-EU_0207/10	R31, R36	477-7552
3	10k	R-EU_R1206	R18, R19, R20	223-2394
4	10n	C-EUC1206	C7, C10, C42, C43	464-6830
1	10μ/10V	CPOL-EUE2-5	C25	228-6622
2	15	R-EU_R1206	R4, R5	223-2013
2	15k	R-EU_R1206	R6, R7	223-2417
1	20MHz	CRYTALHC49U-V	Q5	657-656
6	22p	C-EUC1206	C13, C14, C15, C16, C17, C18	464-6751
8	27k	R-EU_R1206	R12, R13, R16, R17, R39, R40, R43, R44	223-2445
1	32,768kHz	CRYTALTC26V	Q4	226-1443
2	33	R-EU_0207/10	R9, R10	477-7619
1	33p	C-EUC1206	C12	464-6773
2	47k	R-EU_R1206	R22, R25	223-2489
4	47n	C-EUC1206	C5, C6, C38, C39	464-6925

1	47μ	CPOL-EUE2-5	C11	228-6751
2	68/1W	R-EU_0411/3V	R3, R35	131-766
2	74HC573D	74HC573D	IC4, IC5	380-0544
1	100	R-EU_R1206	R34	223-2120
4	100k	R-EU_R1206	R23, R24, R46, R47	223-2524
25	100n	C-EUC1206	C1, C2, C3, C4, C19, C20, C21, C22, C23, C24, C26, C27, C28, C29, C30, C31, C33, C34, C35, C36, C44, C45, C46, C47, C48	464-6852
2	100μ	CPOL-EU153CLV-0807	C9, C41	367-9615
1	220μF/35V	CPOL-EUE3.5-8	C37	228-6773
4	270	R-EU_R1206	R21, R26, R28, R29	223-2170
1	470	R-EU_R1206	R27	223-2215
1	560p	C-EUC0805	C32	211-3316
4	680	R-EU_R1206	R14, R15, R41, R42	223-2237
1	7404SO14	7404SO14	V6	177-6045
2	7432	7432	V4, V5	380-0392
1	Analog	JP3Q	JP7	251-8648
1	Aux Power	NSP-BUCHSE	X3	486-662
2	BC547	BC547	T1, T2	296-087
1	BC557B	BC557B	Q2	348-9456
2	BYG21M	DIODE-DO214AC	D8, D9	995435
1	CS8900	CS8900A	IC3	491-5726
1	CTS0/RXD1	JP2E	JP17	251-8648
1	EIB	JP2Q	JP15	251-8648
2	FFKD2	FFKD2	KNX0, KNX1	101-7015+ 101-7037
4	HCPL2531	HCPL2530	OK9, OK10, OK11, OK12	3598986
1	INT	JP4Q	JP11	251-8648
1	LF33CV	LM340H-05	IC2	355-4128
1	LF50CV	LM340H-05	IC6	355-4257

1	MAX3232	JP4Q	JP1	251-8648+ 251-8503
1	MAX3232CPE	MAX3232CPE	IC1	189-1453
1	MB90F330	MB90F330M21	U$1	/
1	MD1	JP1E	JP5	251-8648
1	MODE1/X1	JP3Q	JP13	251-8648
1	Mode	JP2E	JP6	105-6341
1	P6X_DC	JP2Q	JP8	251-8648
1	P60/P61	JP2QE	JP10	251-8648
1	Power	DIP02YL	S2	103-0743
1	Programming	LED3MM	LED1	228-4979
1	Reset	DTS-3	S1	378-6303
1	SD/MMC	JP2Q	JP9	251-8648
1	SD/MMC	SD_MMC_SOCK	SDCMF-10915W010	4381592
2	SMBJ40CA	DIODE-DO214AC	D1, D7	486-1691
2	TP-UART	TP-UART	TPUART1, TPUART2	/
1	TPUART1	JP4Q	JP2	251-8648
1	TPUART2	JP4Q	JP14	251-8648
1	TXD1/RTS0	JP2E	JP16	251-8648
1	UART0	F09H	X1	259-3330
1	UART1	F09H	X2	259-3330
1	UE-LT1S023A-34	UE-LT1S023A-34	ETH10	/
1	USB	JP6Q	JP3	251-8648
1	USB Mini Host	USB_CONA-V	USB1	418-0194
1	USB Slave	USB_CONB	USB2	458-1648
1	USB_Mode	JP2E	JP4	251-8648

Table 11: KNXcalibur: Part list

A.6 KNXcalibur: Component placement

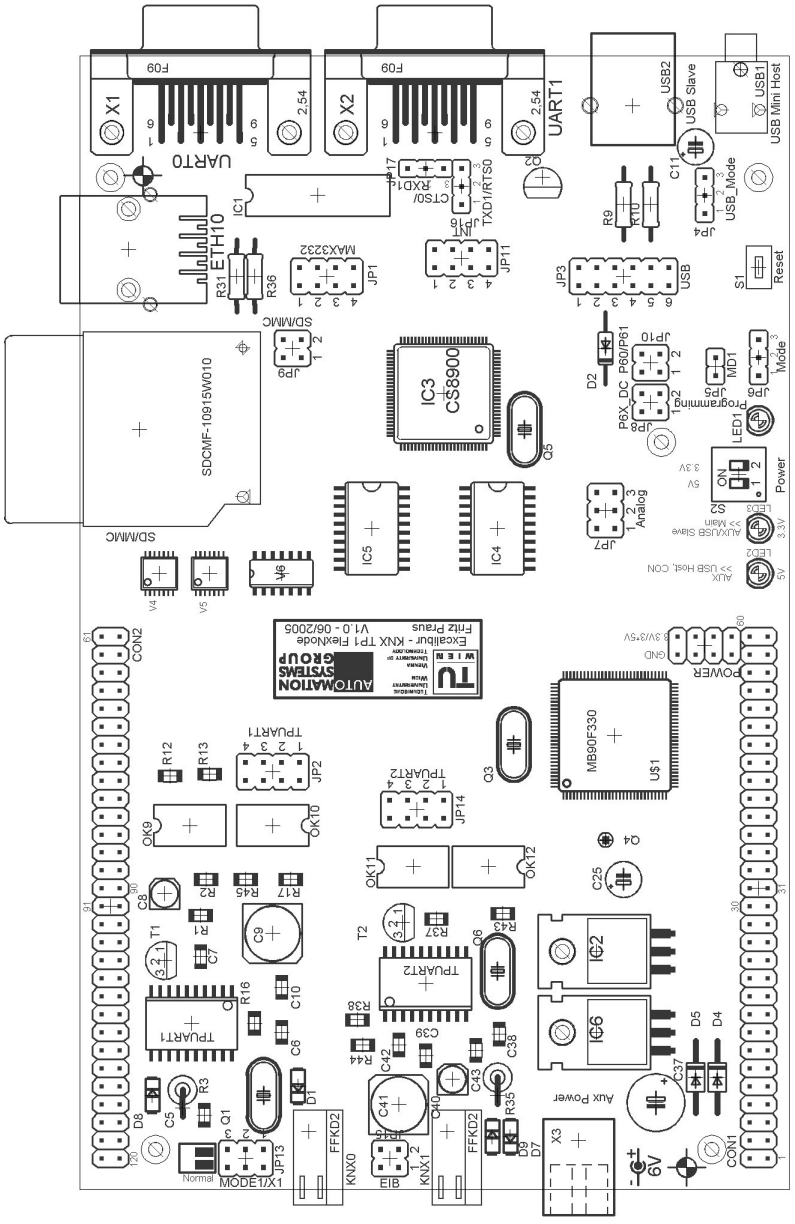


Figure 43: KNXcalibur: Component placement top

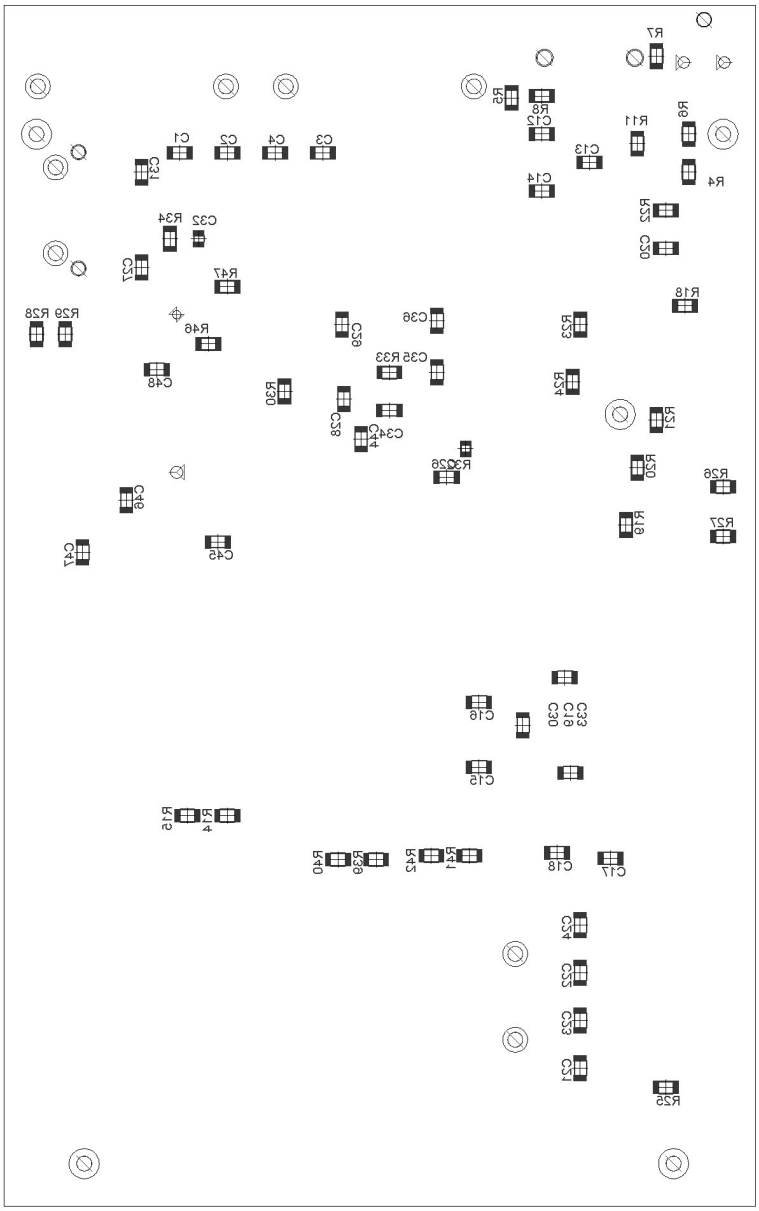


Figure 44: KNXcalibur: Component placement bottom

A.7 KNXcalibur: Board

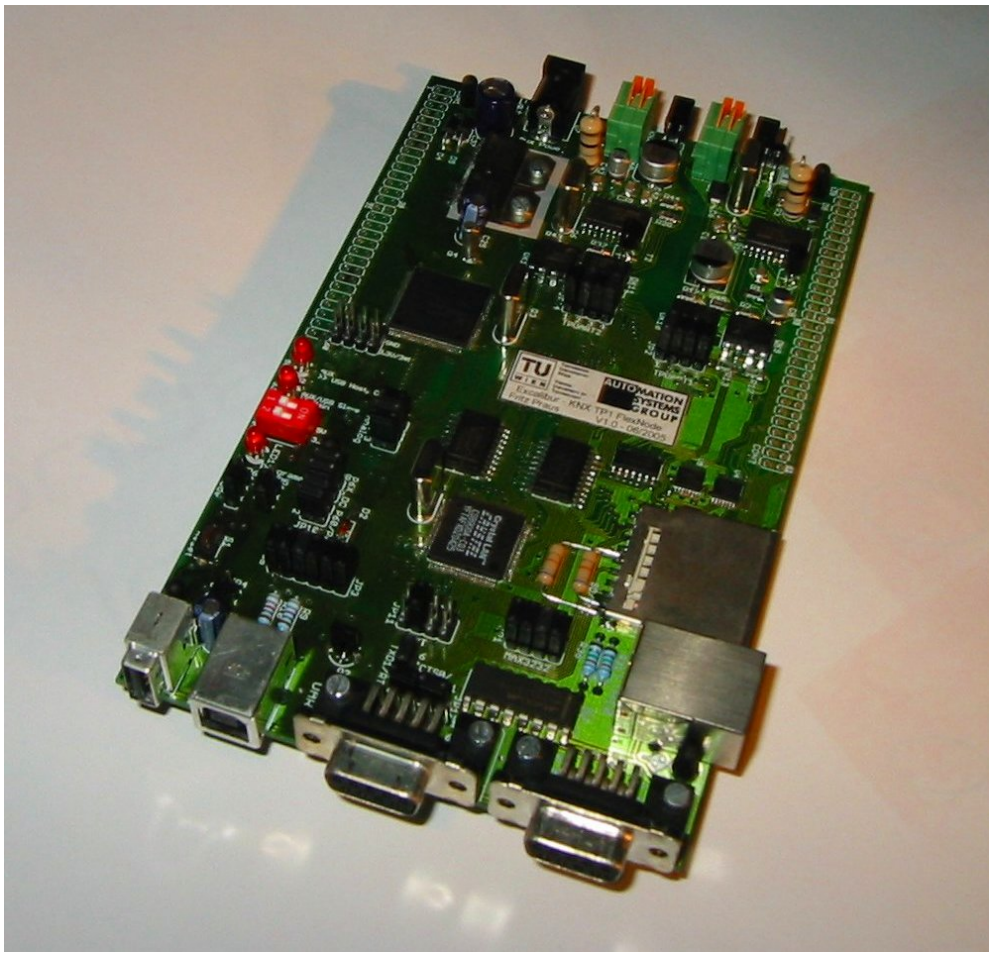


Figure 45: KNXcalibur: Picture

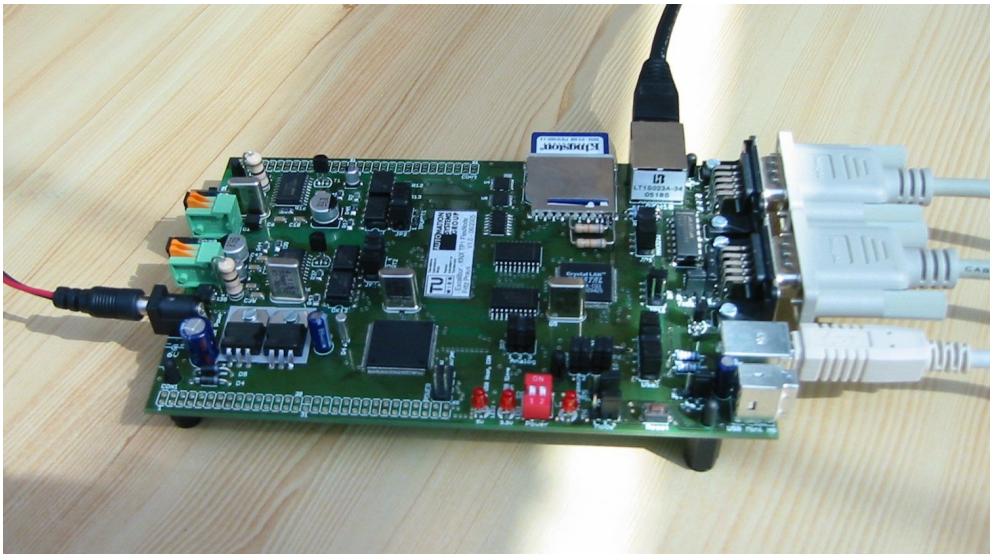


Figure 46: KNXcalibur: Picture