



Abstract

The deployment of building automation systems (BAS) allows to increase comfort, safety and security and reduce operational cost at the same time. Today such systems typically follow a two-layered hierarchical approach. While control networks interconnect distributed sensors, actuators and controllers, a backbone provides the necessary infrastructure for management tasks hosted by configuration and management devices. In addition, devices interconnecting the control network with the backbone on the one hand and the backbone with further networks (e.g., the Internet) on the other hand play a strategic role. According to their particular functionality and the resulting demands on the necessary hardware and software support, BAS devices can be categorized in three distinct device classes. Based upon this classification the paper presents a generic hardware and software framework whose flexible design fits all these classes. Devices for a particular purpose (i.e., class) can be derived from the universally applicable, generic framework, easing development. This process is demonstrated for a multi-purpose hardware platform applicable to multiple device classes. Moreover, three case studies for this platform are presented which illustrate how software applications from all classes make use of this universally applicable platform.

Building Automation Systems

- Three level functional model
 - Field: measuring, moving, switching
 - Automation: (open and closed loop) control
 - Management: recording, archiving, central monitoring
- Two level network topology


 KNX Scientific Conference 2006
 A versatile networked embedded platform for KNX/EIB – 2
 



In [1] a functional model for all kinds of BAS is described. In this model, the system functionality is divided into three levels which are ordered hierarchically. At the field level, environmental data are measured and parameters of the environment are physically controlled. Automatic control is performed at the automation level whereas global configuration and managements tasks are realised at the management level. For years, the levels of this functional model have been mapped to separate networks when BAS were implemented.

Nowadays, the standard three level functional model can be implemented as a flatter, two-level architecture [2]. This is for two reasons. First, so called intelligent field devices can provide more functionality than before – functionality traditionally associated with the automation level. Second, information technology (IT) and its infrastructure became accepted not only at the management level, but also again for portions of the automation level.

As a result, the two-level architecture consists of a control network level and a common backbone which together form the building automation network (BAN). The control network is home to field devices and has a typical bandwidth in the order of a few KBit/s. Since the requirements of management devices still cannot be fulfilled by this control network (e.g., a global consistent view of the entire system needs higher data rates), control networks are interconnected via a high-bandwidth (typically MBit/s) backbone network. At the intersection points between the networks, interconnection devices have their place.

Device Classes

- **Sensors, Actuators and Controllers (SAC)**
 - Interact directly with physical environment
 - Data acquisition
 - Controller functionality
- **Interconnection Devices (ICD)**
 - Link network segments/systems
 - Repeater, bridge, router, gateway
 - No application related function
- **Configuration and Management Devices (CMD)**
 - Configuration and Maintenance
 - Monitoring
 - Logging

 KNX Scientific Conference 2006 
A versatile networked embedded platform for KNX/EIB – 3

Based on this novel view on a BAN topology, BAS devices and their functionality can be re-classified.

Sensors, Actuators and Controllers (SAC) are located at the control level. Representatives of this device class interact directly with the physical environment and are responsible for data acquisition and for controlling the behaviour of the environment. Additionally, they may include controller functionality.

Interconnection Devices (ICD) link different networks and network segments together. Representatives of this device class enable devices from different control network segments to communicate and interact. This can involve simple routing, protocol conversion (gateway) or tunnelling approaches.

Finally, Configuration and Management Devices (CMD) are used to configure and maintain a building automation system. Typical CMDs implement tasks including the change of control parameters, setting up new automation tasks as well as monitoring, logging and archiving process data values.

Goal and Outline

Can we create a modular, generic architecture for all BAS device classes?

- Modular hardware and software architecture
- Prototype implementation: KNXcalibur

**AUTOMATION
SYSTEMS
GROUP**

KNX Scientific Conference 2006
A versatile networked embedded platform for KNX/EIB – 4

**TU
WIEN**

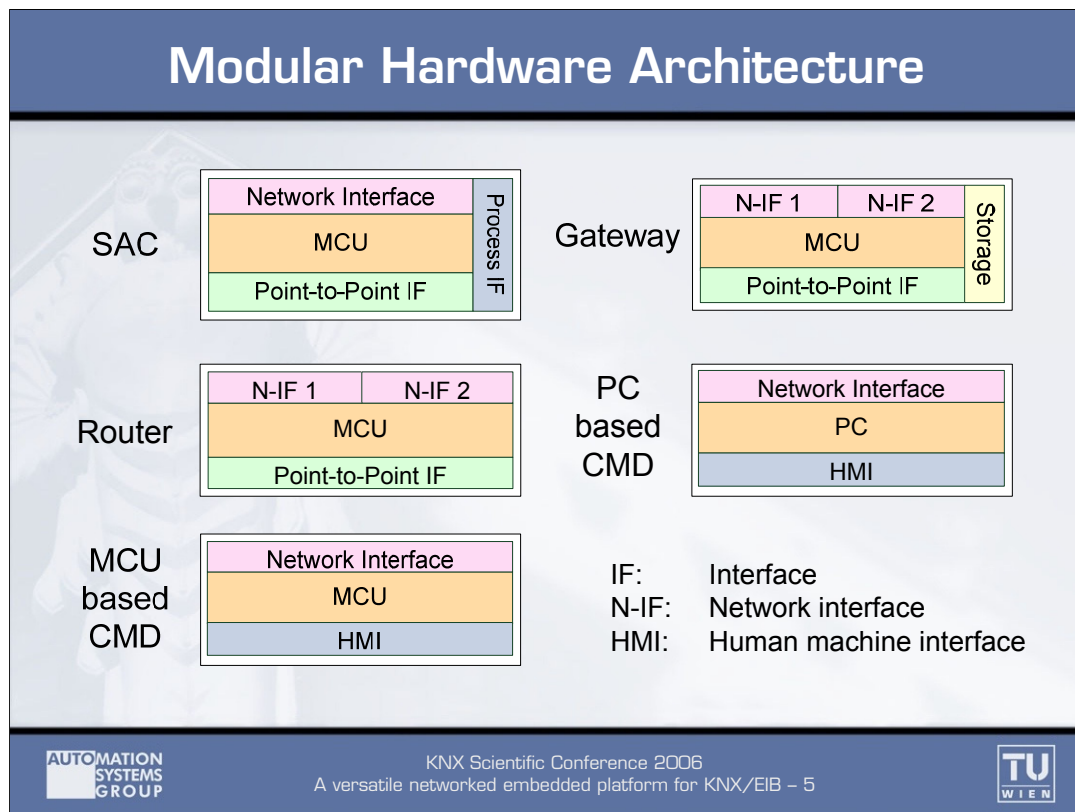
The goal of our work is to simplify the design and implementation of BAS devices. Can we exploit common aspects of their functionality to find a generic hard- and software architecture for all kinds of devices fulfilling embedded control, interconnection, and configuration and management tasks? [5] presents a first requirements analysis for the different device classes with respect to demands on:

- Processing power
- Memory
- Power consumption
- Desirable interfaces

In addition, the requirements analysis includes a discussion of the demands resulting from the installation environment (i.e., where the device is supposed to be located) and of course development and maintenance cost.

Based on this analysis, generic hardware building blocks can be identified which can be assembled to provide the functionality for each device class. The hardware building blocks are: CPU (including RAM), external persistent storage (required only if not already integrated with the CPU, as it is the case with most microcontrollers), power supply (i.e., transformer plus voltage regulator or battery), interfaces (i.e., network interface (IF) for BAN connection, point-to-point IF for configuration tasks, process IF for interacting with the physical environment, or human machine IF for user interaction).

In the following, the modular and generic hardware and software architecture and a prototype implementation named KNXcalibur shall be presented.

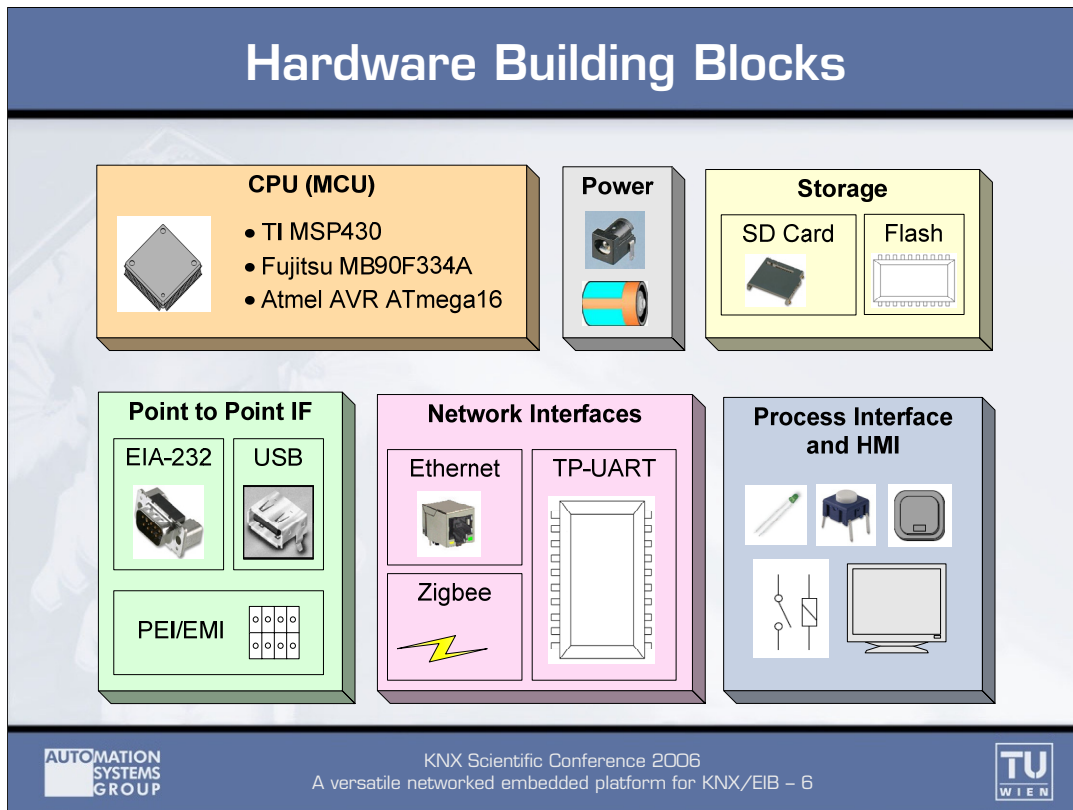


By selecting the corresponding hardware blocks, SAC, ICD or even MCU based CMD devices can be designed.

As SACs have to perform more or less simple tasks, small and low cost 8 or 16 bit microcontroller unit (MCU) based solutions are sufficient. The memory requirements are relaxed since the amount of process data (in the order of bytes to a few KBytes) to handle can be assumed as being small. A SAC device obviously requires a network IF and a process IF and optional a simple point-to-point IF. SACs are often supplied via link power to avoid the need for an additional power cable. In some cases, SACs may be even driven by a battery (e.g., glass break sensors). Here, low power consumption is of major concern. As SACs are usually located in the field, they have to be robust and small.

Compared to SAC devices, the requirements on ICDs are quite similar. However, routers and gateways may need additional memory for storing routing tables or caching data and at least two (possibly different) network IFs. ICDs lack a Process IF. Since battery driven ICDs are uncommon, low power consumption is not as important as it is for SACs. Interconnection devices will normally be located at central points in the building (e.g., in a switch cabinet). However, it is still important that ICDs are small and maintenance-free.

Since CMDs are used to process data collected from the whole BAS, they require more processing power and (persistent) storage for the data to be processed. Obviously they need a Network IF to collect the necessary data as well as an elaborate human-machine IF to interact with the user (e.g., system operator). Since server, workstation and PC-based CMDs are supplied via the power grid, power consumption is not of concern. CMDs with fully fledged UIs will be normally located in mild environments (offices). Therefore, small size and robustness against rough environmental conditions are less important.



Each of the discussed hardware blocks can be realised by a variety of different hardware components. When designing a device for a particular application, appropriate ones are selected to tailor the solution to specific requirements and technologies.

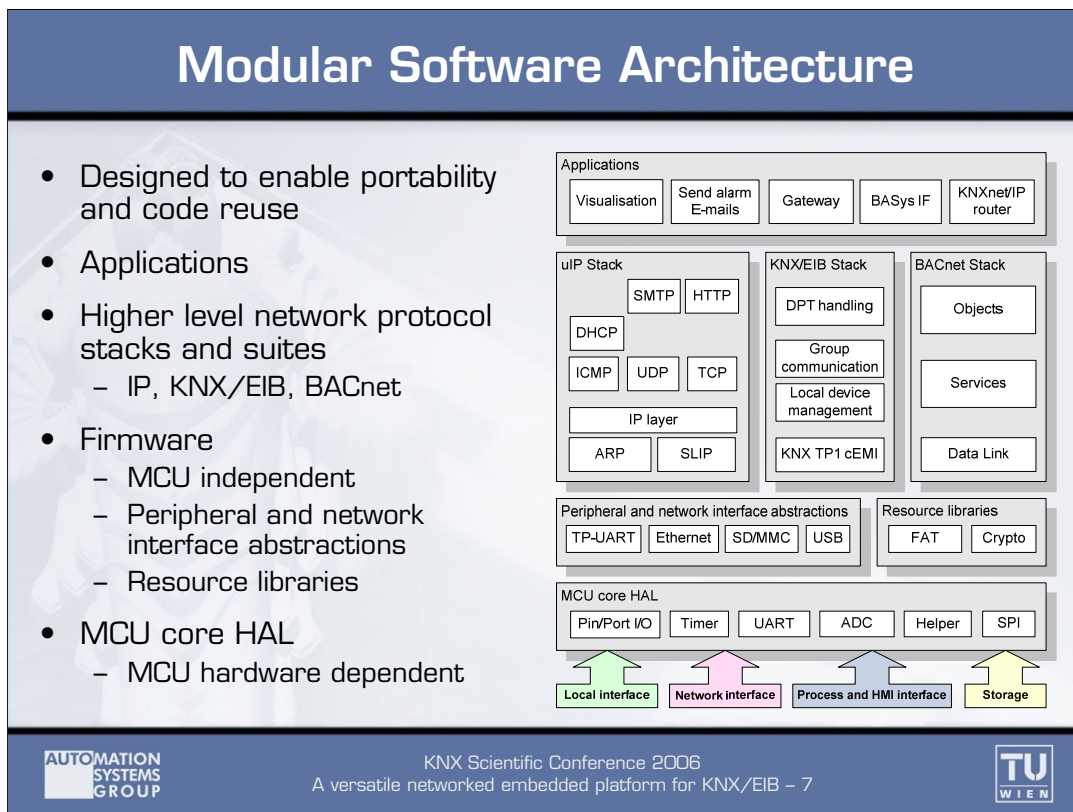
Different CPUs vary in, e.g., performance, cost, availability or power consumption. Obviously, the power consumption and desired lifetime without requiring maintenance (in particular, changing batteries) of a BAS device play an important role in selecting an appropriate power source. For additional persistent storage, an SD/MMC card may be chosen for a removable solution, or soldered-on Flash EEPROMs for a compact design or when MCU program memory needs to be extended. Finally, a few of the many possible peripherals for interfacing with other devices or humans include:

Local IF: EIA-232, USB, or PEI/EMI

Network IF: Ethernet, TP-UART for KNX TP1, or ZigBee for wireless connections

Process IF: typical BAS sensors and actuators

Human Machine Interface (HMI): LEDs, various displays and keypads, touchscreens



The software architecture intends to provide building blocks which can be mixed and matched to support any particular application from any of the device classes. The architecture is designed to maximise code reuse. A change in the combination of the software modules or a change in the hardware design should only require a minimum of modification to the software.

Just as the hardware architecture does for hardware components, the software architecture identifies classes of software modules which are applicable for every device.

At the lowest level, the MCU core hardware abstraction layer (HAL) hides the peculiarities of basic I/O handling and on-chip peripherals. It basically does for them what the C programming language (which is used throughout in this architecture by higher-level modules) does for the MCU instruction set architecture.

This HAL enables MCU independent firmware modules to be written in a highly portable manner. These firmware modules fall into two classes. First, modules which support more complex peripheral ICs such as network or storage controllers; second, entirely hardware independent resource libraries.


Network protocol stacks such as IP (and related protocols), KNX/EIB or BACnet can be integrated according to the requirements of a particular application. If implemented with care, they too are completely hardware independent and can be deployed on different hardware architectures.

Located at the very top of this architecture, applications such as visualisations or gateways make use of APIs provided by the underlying layers.


Although the architecture follows a layered approach, any layer may access the hardware directly if required for performance reasons or special features, at the cost of losing portability.

KNXcalibur: Goals

- Universally applicable with KNX in mind
 - SAC, Router, Network IF for PC-based CMD, MCU-based CMD, Gateway
 - Design openly available
- Hardware
 - Compact, lean and low cost
 - Flexible, extensible
 - Experimental embedded platform (daily lab use)
- Software
 - Open source projects



KNX Scientific Conference 2006
A versatile networked embedded platform for KNX/EIB – 8



Pioneering the presented approach, KNXcalibur [3,4] was developed. KNXcalibur provides a versatile networked embedded platform. It is able to fulfil requirements of all three presented device classes and satisfies the following additional constraints:

It is universally applicable as a gateway and interface for KNX/EIB and serves as a basis for further work in the scope of home and building automation. The design of the hardware and software is openly available.

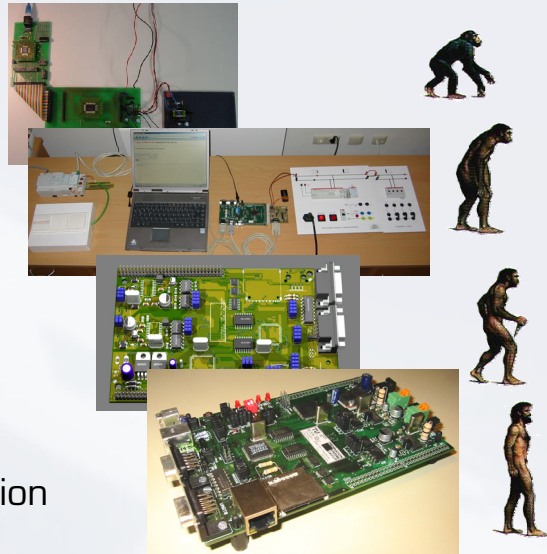
The platform is designed as a compact and low cost stand-alone device. This includes costs for PCB manufacturing as well as costs and availability of used components. It is electrically safe and easy to handle. The used hardware is powerful, meaning that enough processing power and memory are present to implement services like e.g. TCP/IP, USB or a security extension to KNX/EIB like EIBsec. Persistent storage beyond the MCU on-chip flash memory is provided.

The platform is not 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.

KNXcalibur follows the presented software architecture. Existing open source projects are leveraged and new software components again released as open source wherever reasonable.

KNXcalibur: Hardware components

- Fujitsu MB90330F
16-bit Microcontroller
 - 24 MHz, 24 KB RAM, 384 KB flash memory
 - 4 UARTs and USB
 - External bus interface
- Crystal CS8900ACQ3 Ethernet Controller
- 2 TP-UARTs
- SD/MMC card connection



KNXcalibur is based on a Fujitsu 16 bit MB90330 family MCU. It operates at a maximum frequency of 24 MHz and provides 24 Kilobytes RAM, 384 KB flash, 4 UARTs, a 8/10 Bit A/D converter and SPI (Serial Peripheral Interface) as well as an external bus interface similar to the ISA bus. Moreover, USB functionality with device (USB 2.0 full speed) and mini host support is integrated into the controller.

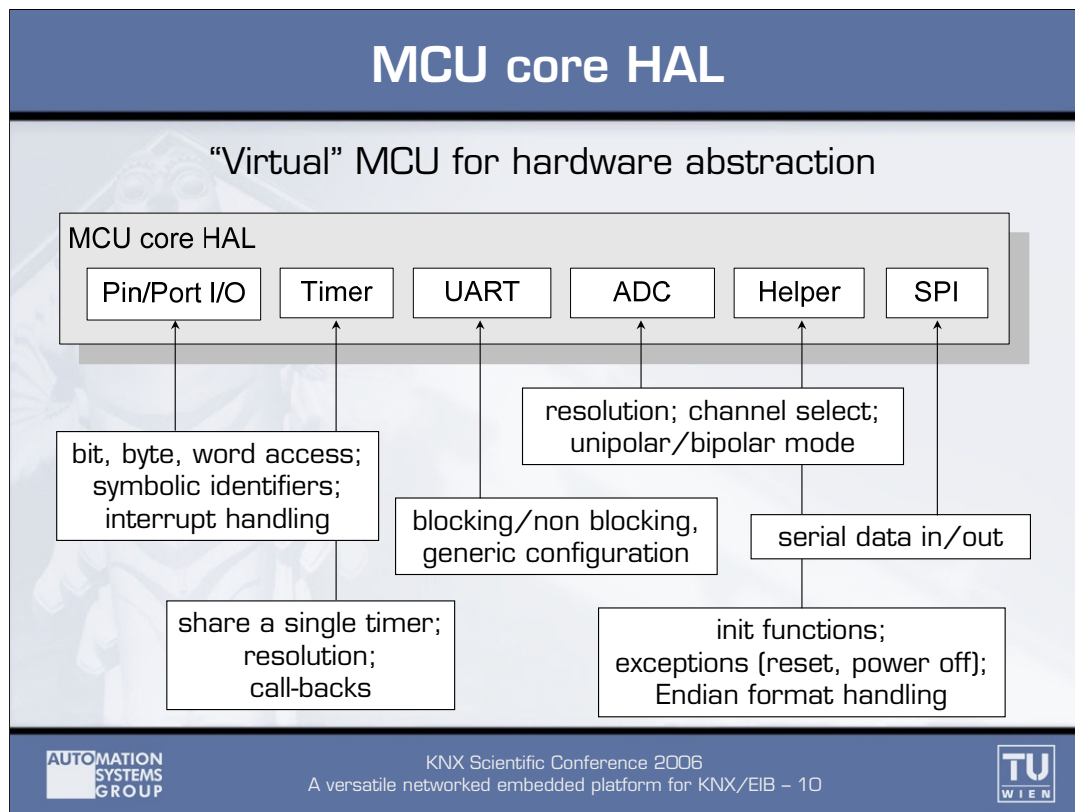
Ethernet connectivity is realised via the Cirrus Logic CS8900A Ethernet LAN controller. It is a single chip, low-cost controller for embedded applications, which supports 10 MBit/s link speed and is accessed via an ISA bus interface.

To support persistent storage of large amounts of data on KNXcalibur without requiring writing to the MCU on chip flash memory and to extend the latter, a SD/MMC card connection has been integrated. The SD/MMC card is accessed via SPI.

For EIA-232 serial connections to the PC side, the design contains true level converters (MAX232) and SUB-D connectors.

Connection to KNX/EIB is realised with the Siemens TP-UART IC providing layer 1 and layer 2 access. It is the easiest, most flexible and cheapest possibility for accessing the KNX/EIB twisted pair medium. Optocouplers are used for galvanic isolation.

All MCU pads are available on pin headers, allowing extension daughterboards to connect. Additional flexibility is provided insofar as hardware blocks (e.g., KNX/EIB IF, Ethernet IF) which are not required for a particular application can be physically disconnected from the MCU, freeing up MCU I/O lines.

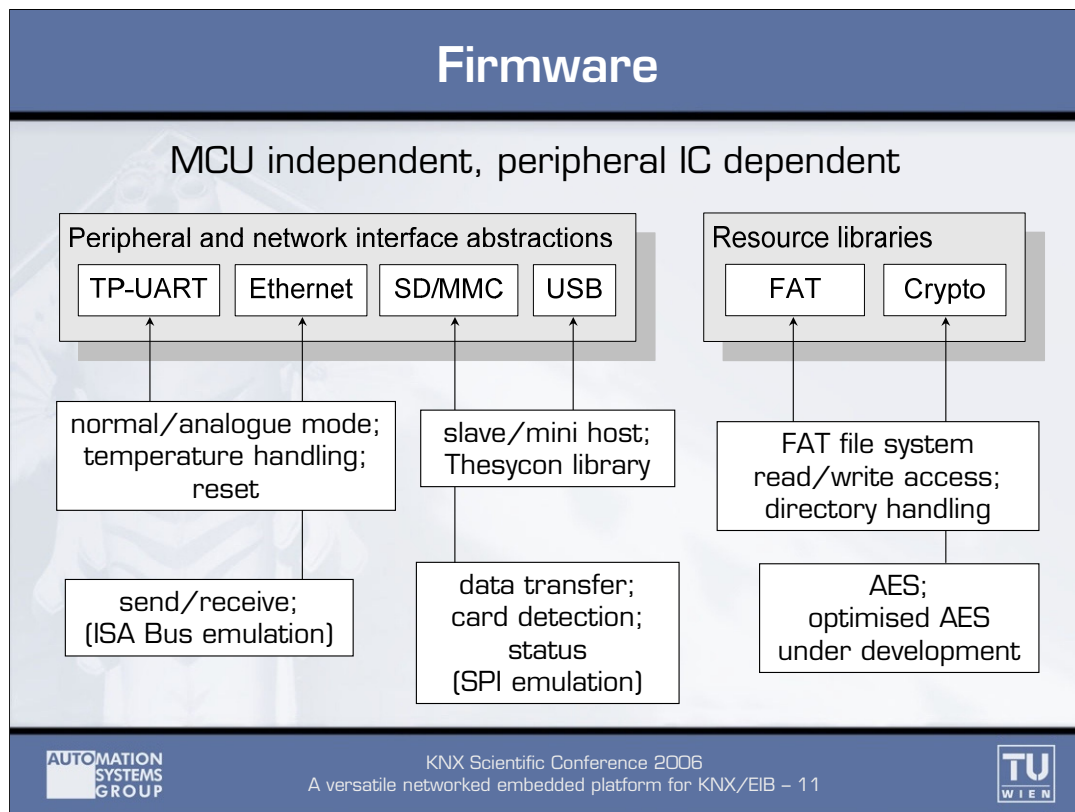


For the development of portable code, the MCU core Hardware Abstraction Layer (HAL) is responsible for the necessary hardware abstraction. The layer creates a kind of “virtual” MCU with core functions, all MCUs must share.

Parts which are not present in hardware may in some cases be emulated in software if the performance hit can be taken. In particular this applies to additional asynchronous and synchronous serial communication interfaces (UART, SPI)– a very basic application of Hardware/Software Co-Design principles.

The provided APIs are split into the following parts:

- Pin/Port I/O API: bit-wise, byte-wise and word-wise access via a single instruction; configuration (e.g., data direction, polarity, pattern polarity, pattern transition); interrupt handling
- Timer API: allows applications to share a single hardware timer; configuration (e.g., resolution, continuous cycle/single shot operation, timer constant register); call-backs are provided
- UART API: sending and receiving of data in blocking or non-blocking mode; configuration (e.g., start/stop bit, parity, baud rate)
- SPI API: communication via the Serial Peripheral Interface (serial data out, serial data in); configuration
- ADC API: currently under construction – will provide different modes (unipolar/bipolar), channel select, resolution, ...
- Helper API: for hardware dependent functions (e.g., main configuration, initialisation, and exception handling, Big-Endian/Little-Endian format handling)



AUTOMATION SYSTEMS GROUP

KNX Scientific Conference 2006
A versatile networked embedded platform for KNX/EIB – 11

TU WIEN

Resting upon the MCU core Hardware Abstraction Layer, the peripheral and network interface abstractions provide hardware independent access to the underlying peripheral ICs.

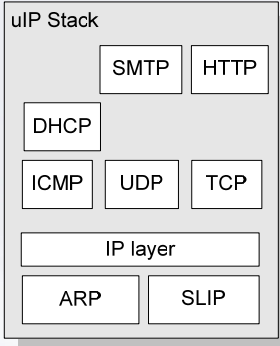
- TP-UART API (relying on the UART and Pin/Port I/O API): asynchronous serial communication in normal and analogue (transceiver only) mode
- Ethernet API: device driver for the Ethernet controller – sending and receiving of Ethernet frames, checksum calculations; configuration (MAC address, auto negotiation of link speed). For KNXcalibur, an emulation of the ISA bus, DOS like access functions and connection to the CS8900A is handled by this module. The ISA bus emulation was considered too application specific to be included in the core HAL.
- SD/MMC card storage API: data transfer, card insertion and removal. On KNXcalibur, the SD/MMC card is accessed via software SPI to keep the MCU pins assigned to the hardware SPI available for their alternate function, i.e., as external interrupt lines. For the time being, the software SPI emulation is still integrated with the SD/MMC card API module for performance reasons.
- USB API: support for slave/mini host capabilities, full and low speed data transfer (control, bulk and interrupt), covers USB enumeration, provides USB events to the application (e.g., attach and remove). For KNXcalibur, the Thesycon FUMA - Fujitsu USB Minihost API [6] is used. This API exposes almost no specifics of the USB controller and thus can serve as a good basis for a truly generic abstraction.



Resource libraries are entirely hardware independent; they implement functionality that is either purely algorithmic or already defined upon a hardware abstraction.

- FAT (File Allocation Table) file system API: FAT-16 (future: FAT-12, FAT-32) read/write access, open/create a file/directory
- Crypto API: currently under construction – will provide an optimised version of the Rijndael (AES) reference implementation

IP Stack

- Open source project μ IP [7], well documented
- Small code size and low RAM usage, adjustable at compile time
- IP via Ethernet or EIA-232 (ARP, SLIP)
- Fully RFC compliant IP/UDP/TCP implementations; ICMP echo (Ping)
- Web server, SMTP client (E-mail), Telnet server, DNS resolution, DHCP




 KNX Scientific Conference 2006
 A versatile networked embedded platform for KNX/EIB – 12
 

For providing the TCP/IP stack required by many applications, the open source project μ IP [7] was selected. μ IP is a TCP/IP protocol stack designed for 8-bit and 16-bit microcontrollers which features a very small code footprint and RAM requirements.

It implements all protocols of the IP suite which are essential for Internet communication. IP, UDP, and TCP are provided. The number of concurrent TCP connections (both active and passively listening) is only limited by available RAM. Also, μ IP answers ICMP echo requests (Ping). ARP for IP over Ethernet is available as well as SLIP. μ IP supports a single network interface at a time.

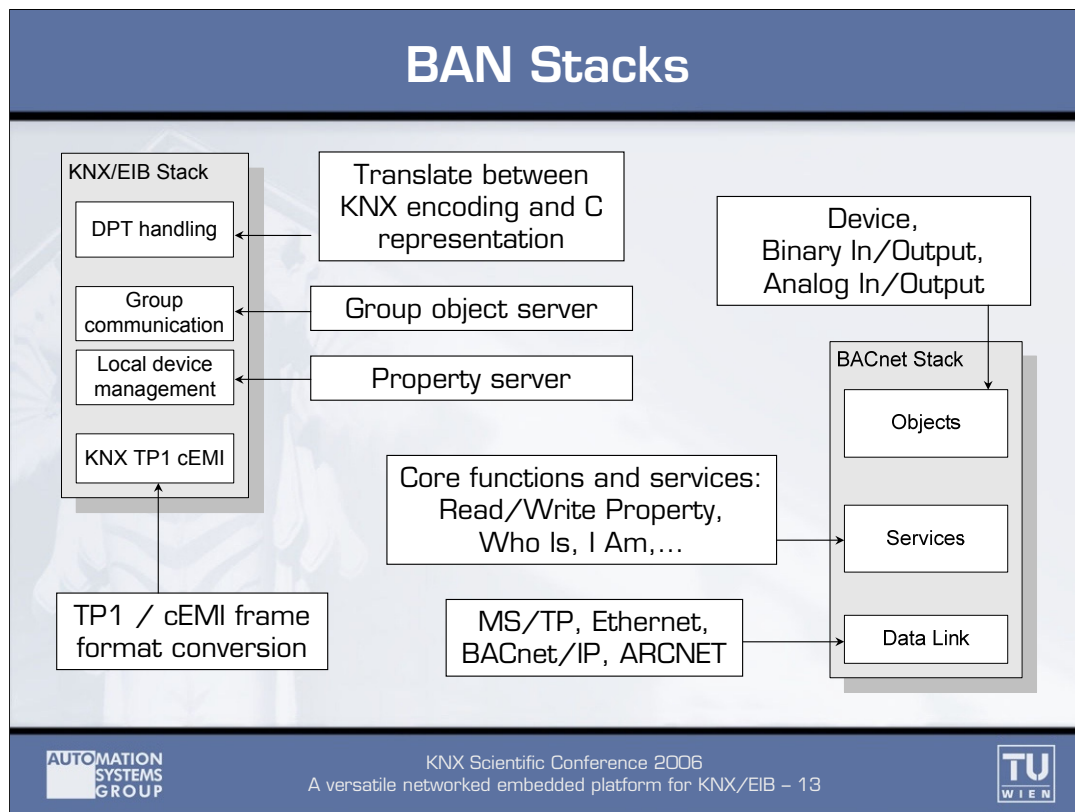
Unlike many other TCP/IP implementations for small systems, μ IP is fully RFC compliant. All requirements regarding host-to-host communication are implemented, making devices first-class network citizens. IP fragmentation is also supported.

The code size is on the order of 10 kilobytes and RAM usage can be configured to be as low as a few hundred bytes. Both figures depend on which protocols are compiled in. The RAM requirements can additionally be controlled by adjusting certain operation parameters, e.g., setting the maximum number of TCP connections.

The μ IP stack includes various demo applications which provide basic implementations of higher level IP suite protocols such as a simple HTTP (Web) server, an SMTP E-Mail client, a Telnet server, a DNS resolver and a DHCP client.

Many ports to different microcontroller architectures and Ethernet controllers are already available. For KNXcalibur (i.e., the Fujitsu FPMC-16 architecture) however, no such port existed yet and therefore had to be undertaken in the course of the present project. The necessary device driver for the Ethernet controller is provided by the Ethernet module presented on the previous slide.

μ IP uses a custom library for extremely lightweight threads (“Protothreads”). They do only require a handful of RAM bytes per thread, but no separate stacks. The ability to use conditional blocking inside C functions facilitates the programming of event driven systems. Thus, the Protothreads library is also useful outside the context of μ IP.



For connecting to different BANs, two stack implementations are currently under development. Essential functionality is already available.

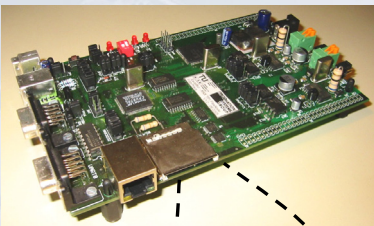

The KNX/EIB stack relies on the cEMI message format.

- The KNX TP1/cEMI link module uses the TP-UART API. To its clients, it provides the cEMI L_Data link layer services. This includes conversion between the respective frame formats. The current implementation is limited to TP1 standard frames. Interface object property access from the KNX network (via TP-UART) is routed to the local device management module.
- The local device management module implements a persistent store for device configuration properties. It implements the cEMI property services for use both by the KNX link module and local applications (including an EIBnet/IP device management server).
- The group communication module provides a basic group object server both for use by the local application as well as read requests from the KNX network. It manages group address association and masks the PCI encoding/decoding from the local application.
- The DPT handler is used by local applications to handle the encoding of the KNX ASDUs provided by the group object and property servers.

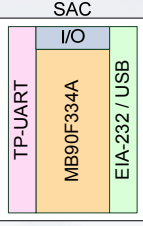
The BACnet stack relies on an open source project [8] which was ported to the KNXcalibur platform. The project provides an implementation of BACnet application layer, network layer and MAC layer communications services tailored to embedded systems. The data link layer provides support for BACnet/IP, MS/TP, ARCNET and Ethernet. KNXcalibur currently uses Ethernet as the transport medium.


Implementation of the various core functions and services to be provided by the stack is still in progress. However, the most relevant services (e.g., Who-Is, I-Am, Who-Has, I-Have, Read Property or Write Property) are already available. The stack also includes some example objects, including the essential device object. Properly configuring it enables support of the I-Am and I-Have service procedures as well as rejecting unsupported service requests.

KNXcalibur as SAC device: “USB key” door opener





- Thesycon FUMA USB firmware
 - Detect USB Mass Storage
 - Configure bulk endpoints
- SCSI read10 command
 - Block access to data on USB stick
- FAT16 file system
 - Read MBR, VBR, directories and keyfile





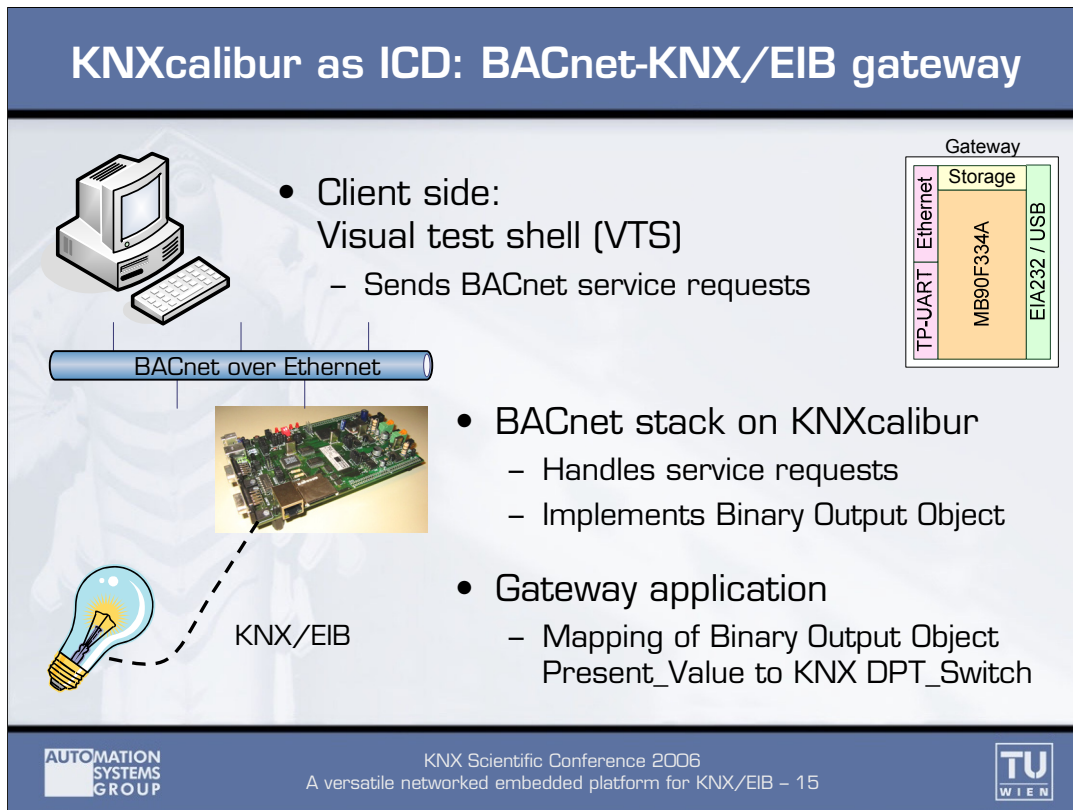
KNX Scientific Conference 2006
A versatile networked embedded platform for KNX/EIB – 14



The universally applicable and flexible design of KNXcalibur is now presented in case studies. Each case study relies on an instantiation of the modular hardware architecture with the presented building blocks and necessary parts of the introduced software framework.

The first application is an “USB key” door-opener as a representative of a SAC device. A key (64 bit) located on a standard USB Flash memory pen drive is used for authentication. After the USB connection has been made, KNXcalibur reads the keyfile and compares it against its list of valid codes. If the keys match, appropriate actions are taken via the local process interface (I/O ports) or the KNX network to, e.g., open a door. It should be noted that the focus here is on providing a proof-of-concept for the USB functionality, not a perfect solution from the security point of view.

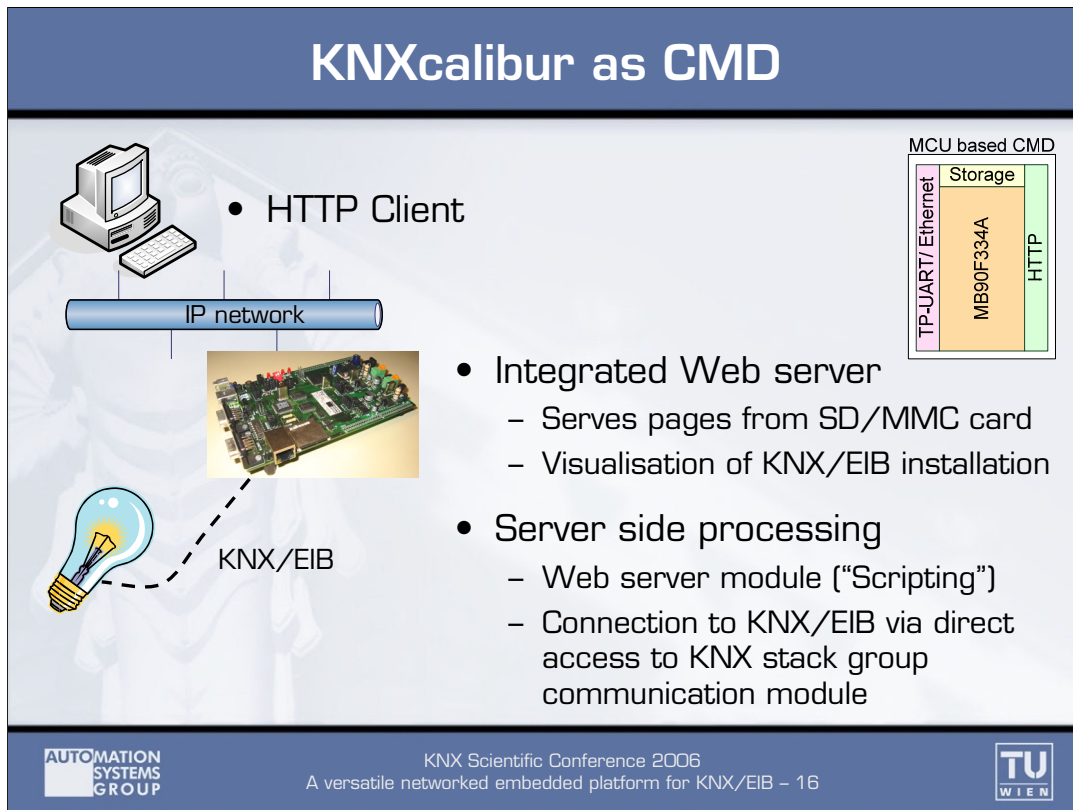
The application is notified by the Thesycon FUMA (Fujitsu USB mini-host API) when a USB device is plugged in. It then uses the FUMA to retrieve the appropriate descriptors to validate if the device belongs to the USB Mass Storage class and configure the bulk endpoints for incoming and outgoing data transfer. The SCSI read10 command is then used to perform a block access (read10) on the Flash drive. Finally, the master boot record (MBR), volume boot record (VBR), directory and the keyfile are read with the help of the FAT16 file system library.



The second case study is a simple BACnet-KNX/EIB gateway as a representative of an ICD. The application currently is limited to propagating changes to the Present_Value property of a BACnet Binary Output Object to a KNX/EIB DPT_Switch (Boolean) datapoint.

On the BACnet side, Ethernet is used as the network interface. The BACnet stack on KNXcalibur is used to implement a Device with a single Binary Output Object. When a WriteProperty service request is received, an appropriate GroupValue_Write request is issued on the KNX TP1 network. ReadProperty service requests currently return the last state written, but the application could easily be extended to obtain the response data via the KNX GroupValue_Read service.

The application was tested against the BACnet Visual Test Shell (VTS). The VTS is an open-source Windows application for BACnet conformance testing. Services tested using VTS include device and object discovery (Who-Is, Who-Has), and property access (ReadProperty, WriteProperty).



As an example for a CMD, an integrated Web server was selected. Such an approach allows to visualize and interact with a KNX/EIB installation via a Web browser.

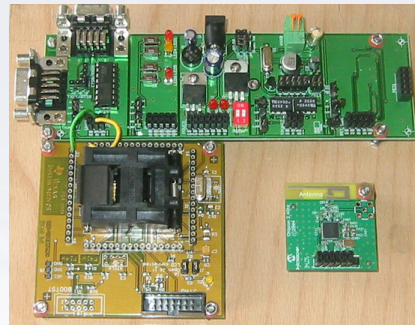
Two approaches are currently under development. In both cases, KNXcalibur is serving web pages located on the SD/MMC card. However, they differ in how the page content is dynamically adapted to reflect the state of the KNX/EIB installation.

- Client side processing: This approach transfers the workload of the visualisation and page generation to the client side as much as possible. The served page basically contains only the available group addresses, associated DPTs and graphic elements for visualization. The application logic to dynamically update the page contents upon a user request or KNX network event is contained in JavaScript code, which is executed in the browser. The JavaScript code interacts with the KNX network via KNXnet/IP Routing by way of a Java Applet. This KNXnet/IP Routing server is also hosted by KNXcalibur.

- Server side processing: In this more traditional approach, a custom Web server module dynamically generates the page content. It interacts with the KNX network by accessing the group communication module (and DPT handling service) of the local KNX stack.

Outlook

- Refine hardware abstraction
- Support new hardware components
 - TI MSP430 (low power), Atmel AVR
 - EIA-485, ZigBee, Bluetooth
 - Appropriate PCB modules →→
- Improve network stacks
- New applications



The first hardware prototype that follows the modular approach presented has been completed. KNXcalibur now serves its daily use in the lab. Currently work is underway on refining the hardware abstraction modules both with regard to interfaces and implementation. Also, support for new hardware components is envisaged.

On the software side, this means porting the MCU HAL to support additional MCUs. On the one hand, support for the Texas Instruments MSP430 MCU is currently being added to provide a very low power platform for battery driven devices. On the other hand, a very low cost Atmel AVR MCU is in mind.

Regarding additional peripheral and network interfaces, EIA-485 for multi-drop serial communication (e.g., for BACnet MS/TP), a Chipcon ZigBee module for medium range wireless connections and Bluetooth for higher bandwidth, low range wireless communication are of particular interest. If required by further components, ISA bus emulation support may also be moved into a software module of its own.

As for the corresponding hardware setup, the image shows our “header board” with electrically separate blocks for EIA-232, a minimal HMI (LEDs, buttons), power supply, TP-UART and a PEI interface. This board can be combined with any MC prototype board, supporting the ongoing development regarding additional MCUs. KNXcalibur extension daughter boards are also being created.

On higher levels of the software framework, the KNX and BACnet stacks still provide ample room for improvement. Also, the collaboration (in terms of control and data flow) between the individual software modules requires further attention. The use of an embedded operating system, maybe also only a lightweight thread library such as Protothreads, could prove beneficial and should be investigated. Of course, new top-level applications are in mind as well.

Acknowledgements to...

- Wolfgang Granzer
for useful tips and comments
- Rainer Müller and Michael Rupprechter
for the “USB key” implementation
- Jürgen Weidinger
for the UART API and BACnet implementation
- Arthur Bliem and Valentin Ecker
for the Web server implementation
- Jörg Rohringer
for the timer API and TP-UART driver implementation

References

- [1] ISO 16484-2. “Building Automation and Control Systems (BACS) – Part 2: Hardware”, 2004.
- [2] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman. “Communication Systems for Building Automation and Control”, Proceedings of the IEEE, vol. 93, no. 6, pp. 1178–1203, June 2005.
- [3] Friedrich Praus, Wolfgang Kastner, and Oliver Alt. “Yet another all-purpose EIBNet/IP gateway”. In Proc. Konnex Scientific Conference, October 2004.
- [4] Friedrich Praus. “A versatile networked embedded platform for KNX/EIB”. Master's thesis, Vienna University of Technology, 2005.
- [5] Wolfgang Granzer, Wolfgang Kastner, Georg Neugschwandtner, and Friedrich Praus. “A Modular Architecture for Building Automation Systems”. Proc. IEEE International Workshop on Factory Communication Systems, June 2006.
- [6] Thesycon FUMA - Fujitsu USB Minihost library:
http://www.thesycon.de/deu/prod_usbfirmware.shtml
- [7] Adam Dunkels. "Full TCP/IP for 8-Bit Architectures". Proceedings of the first International Conference on Mobile Applications, Systems and Services (MOBISYS 2003), May 2003.
- [8] Open source BACnet stack: <http://bacnet.sf.net/>