Smart Sensors and Actors with BACnet™ and Mbed OS on Cortex-M Microcontrollers

Christian Bock
RG CEA, Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Wismar, Germany
christian.bock@hs-wismar.de

Mathias Marquardt
RG CEA, Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Wismar, Germany
mathias.marquardt@hs-wismar.de

Alexander Martens
RG CEA, Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Wismar, Germany
alexander.martens@hs-wismar.de

Olaf Simanski
RG CEA, Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Wismar, Germany
olaf.simanski@hs-wismar.de

Olaf Hagendorf
RG CEA, Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Wismar, Germany
olaf.hagendorf@hs-wismar.de

***Abstract*—The increasing use of intelligent sensors and actors in automation proportionally increases the need for suitable interfaces to either interact with, or to provide the means, to participate alongside a common interface. Without appropriate system prerequisites, the integration of sensors or actors, not equipped with such interfaces, into an Ethernet based network, is not possible. As part of a R&D (research and development) project of the University of Wismar, a modified open-source BACnet™[1] (Building Automation and Control Networks) stack is used as protocol stack for device-to-device communication between sensors and actors. As hardware, Mbed OS compatible ARM Cortex-M based microcontrollers are used. The target is a, by protocol supported, device-to-device communication for virtually any sensor or actor. The primary tasks of devices utilizing a modified embedded BACnet stack therefore are the extension or provision of Ethernet communication abilities via BACnet/IP. The flexibility of BACnet to project virtually any source of information as a group of standardized objects and services allows integrating a wide range of individual devices into Ethernet networks via a common interface. Combining an established and well-proven, royalty-free protocol with a microcontroller operating system provides all necessary means for a universal interface for the integration of "things" into Ethernet based IP networks via BACnet/IP.**

Keywords—BACnet, Mbed OS, Cortex-M, C/C++, open-source

I. INTRODUCTION

The integration of sensors or actors in an Ethernet based IP network, requires not only a compatible network interface, but also standardized means to present or receive information. The BACnet (Building Automation and Control Networks) protocol is a widespread and well-proven protocol, representing devices and their characteristics as objects, either specialized for specific tasks, such as a trend log, or general-purpose objects, e.g. *AnalogValue* object. Additionally, BACnet services, e.g. *Change of Value* notifications or *Alarm* services complement the devices functionality in terms of event based, non-deterministic features. The combination of an embedded device, open-source BACnet stack and Mbed OS, an open-source, connected embedded RTOS (Real Time Operating System) for ARM Cortex-M microprocessors enables the development of highly adaptable gateways. It provides native BACnet communication on one hand, as well as device specific interfaces to, e.g. I²C sensory or proprietary bus interfaces on the other.

II. BACNET

A. The protocol

BACnet, originally developed by ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning engineers) in 1987, is defined as a standard by the ANSI/ASHRAE 135 (since 1995) and ISO 16484-5 (since 2003) standards as a protocol for building automation purposes at process control level. Apart from its origin as process control protocol, BACnet is fully capable of implementing a device-to-device communication network. Therefore, BACnet supports to run a complete automation network in the sense of individual devices, or "things", delivering and receiving information, either as single or multi-target communication. This capability enables the implementation of BACnet as a communication tool for smart devices. The underlying principle of BACnet is to ensure interoperability between devices of different vendors. This is achieved by the definition of a standardized

[1] BACnet™, further referred to as BACnet, is a registered trademark of ASHRAE

TBI Technologie-Beratungs-Institut GmbH TBI-V-1-135-VBW-048

representation of devices partaking in a BACnet network as a sum of objects and services, further detailed by object properties. For projects over spanning multiple vendors, supported object types and properties as much as services of BACnet devices need to be defined in a BIBB (BACnet Interoperability Building Block) to guarantee interoperability between different vendors devices. Additionally, the PICS (Protocol Implementation Conformance Statement) accompanying any BACnet compliant device, describes the capabilities and supported features of the described device, as a reference for design and layout decisions.

B. Physical Layer

The BACnet protocol specifies a four-layer model, spanning all seven layers of the OSI (Open Systems Interconnection model) model [1], [2]. Combining Layers 3 to 6 as one, the BACnet stack implements the *BACnet Network Layer*. The primary task of this layer is consistency, regardless of the actual physical layer underneath, the decoding and management of BACnet messages and service requests from preceding layers. Depending on the physical network to be used for transmissions, Layer 1 and 2 provide the necessary, network-specific adaptions. Fig. 1 shows an adaption, including physical layer specific modifications for a BACnet implementation on layer 1 and 2. The tasks fulfilled by the adapted BACnet layers do not differ from tasks of OSI-Model defined Layers, but are amended in Layer 2, the VMAC (BACnet Virtual MAC Layer) layer. This Layer provides virtual MAC addresses. A virtual MAC address in the sense of BACnet/IP is the combination of IP-Address and UDP port as a pseudo-MAC address. This virtual MAC is used by BACnet as internal MAC substitute address for any BACnet datagram over UDP later on. This allows for connectionless delivery mechanisms, capable of directing information to multiple destinations and ports. This also applies for multiple ports on a single IP-target. Other possible data link layer specified by the BACnet protocol [4] are:

- BACnet ARCnet (Network or EIA-485)
- BACnet Ethernet (IEEE 802.3)
- MS/TP (Master-Slave / Token Passing) on EIA 485
- BACnet over LonTalk
- BACnet PTP (point-to-point) on EIA-232

The remaining layer 7 represents the actual application build upon the BACnet stack.

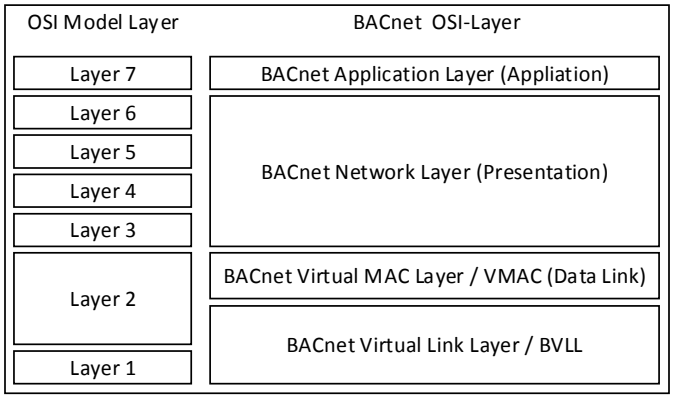

Fig. 1. BACnet OSI-Model network Layer

C. Datapoints and services

The BACnet standard distinguishes between services and objects. Services such as *Alarms* or *Change of Value* notifications are based on events and provide the ability to implement the exchange of information between devices without an explicit prior request. BACnet objects represent a devices data points. They are not scalar values, but structures whose information are represented by properties. Further, BACnet objects are divided into groups by specific purposes, such as e.g. *Analog*- or *Binary-Inputs*, -*Outputs* or -*Values*. Additionally, the BACnet standard also defines objects that are more complex. These do represent e.g. calendars, trend logs or other features specific to a supported field of activity. ID and type are mandatory properties of each object. The object ID has to be unique in each group of objects, e.g. analog objects. Together these two values are unique for each BACnet device, used for object identification. Therefore, the combination of those properties represents a BACnet *Object Identifier*. As a representation of the BACnet device itself, the *deviceObject* is a required object. Apart from information such as vendor details or device descriptions, the *deviceObject* also holds the device ID. The device ID is an internetwork-wide, unique 22bit long unsigned integer, identifying a single BACnet object, independent of data link layer specifics, such as IP-Addresses.

D. Datapoint access

Apart from event-based services, accessing object information is handled by read- and *writeProperty* service requests. With a *read*- or *write propertyMultiple* request, multiple object properties can be accessed in one call. This applies to properties and whole objects. For example, to identify an object as a whole a *read propertyMultiple* is requested and receives a structured-object-list or a simpler object-list, dependent on implementation.

E. Internetwork device identification

The BACnet standard defines 32 services, classified into five groups [4]. The group of *Remote Device Management Services* defines services used to remote control or instruct BACnet devices, individually or as group of devices with device- or object properties, meeting specified conditions. Two of those services, and their respective affirmation response, are of particular relevance for mapping a BACnet network.

- *Who-Is & I-Am*
- *Who-Has & I-Have*

Who-Is asks for devices whose device ID reside within a specific range of device IDs, while *Who-Has* asks for devices providing objects defined by object ID and -type. In both cases, the affirmation service results in one or more device IDs identifying devices meeting set conditions. These services in conjunction with their respective affirmation enable each individual device to locate and communicate with other devices of the same network section, regardless of a master- or control server's presence.

III. MBED OS

A. Introduction

Mbed OS is an open-source operating system designed for ARM Cortex-M microcontrollers [6]. It provides features for IoT (Internet of Things) devices, numerous OO APIs (object-oriented Application Programming Interface) as much as a OO HAL (object-oriented Hardware Abstraction Layer) and out-of-the-box protocol stack implementations. One being the LWIP (Lightweight IP) stack [7], [8]. Fig. 2 depicts an excerpt of some of Mbed OS's features and its structure. Composed of the components "Core", "Runtime" and "IP connectivity", Mbed OS is depicted in the center part of Fig. 2. Those components do not represent the full extent of Mbed OS, but the most relevant parts for the implementation of a protocol stack atop Mbed OS. At the bottom, actual hardware interfaces grant access to IPs (Integrated Peripheral) or IC (Integrated Component) components via e.g. physical bus systems.

- Mbed OS Core

  Apart from different libraries, providing several interfaces regarding storage such as FAT or Little FS file systems, this component provides the necessary modifications for different compiler options (GCC, ARM, IAR).

- Mbed OS Runtime

  The Runtime component of Mbed OS joins Mbed OS's real time environment components (Events, RTX, etc.) with manufacturer components and ports. HAL (Hardware Abstraction Layer) drivers, platform specific MCU SDKs (Micro Controller Unit Software Development Kit) and CMSIS (Cortex Microcontroller Software Interface Standard) provide the essential flexibility and compatibility for Mbed OS to interact with ARM Cortex-M controllers.

- Mbed OS IP connectivity

  This group of stacks and interface drivers, provide the means to establish connections with different network architectures. At the top of this are stacks, e.g. LWIP. Opposite, are manufacturer specific drivers, interfacing physical layer hardware.

B. Mbed OS project deployment

Mbed OS is an open-source embedded operating system [6] whose design goal is the compatibility with a range of different ARM Cortex-M microcontrollers. At its current release, Mbed OS supports 298 platforms of multiple vendors [6]. To handle this multitude of configuration possibilities, a device and target specific deployment solution is provided. Mbed OS integrates *mbed-cli* (mbed command line interface) for target specific deployment solutions. The mechanisms of *mbed-cli* include a system of JSON (JavaScript Object Notation) script files to define and describe extension projects or libraries and several Python tools. This allows to generate projects with predefined configuration variables and target- or platform dependent configurations. In case of BACnet4MbedOS (see IV), this relates to configuration parameters that translate into C/C++

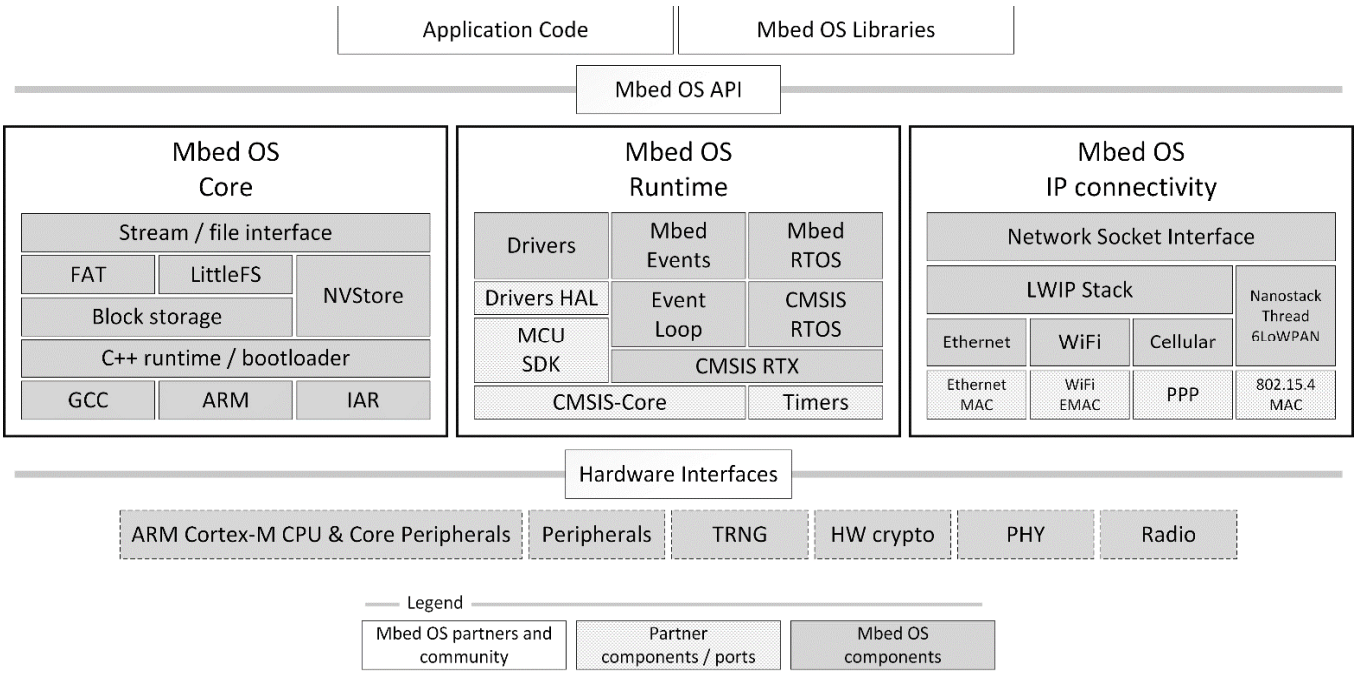

Fig. 2. Mbed OS structure and components

defines or macros. Such as supported object types or physical layer configuration.

C. Mbed OS EventQueue

One potential difficulty in the programming of multi-threaded microcontrollers is the deferral of tasks between threads of different priority. This becomes relevant if e.g. an ISR (Interrupt Service Routine) executes a function, not eligible to run in ISR context. In such cases (see Fig. 3), or to defer a task from high to low priority context, Mbed OS provides an *EventQueue* class. Deferred function calls will be enqueued following a FiFo (First in - First out) principle. This ensures a chronologically consistent execution of deferred tasks. BACnet4MbedOS utilizes *EventQueues* to, e.g. execute callback code invoked due to service requests (see IV.D).

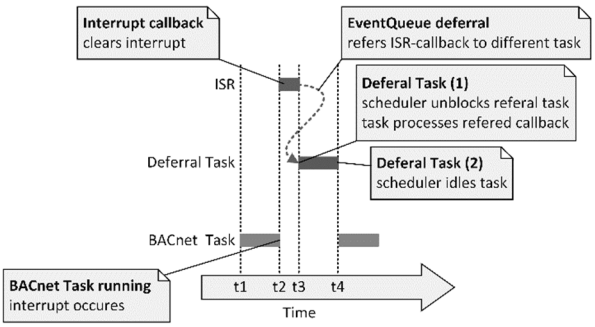

Fig. 3. Task deferral from Thread to Thread

IV. BACnet4MbedOS

A. BACnet embedded port requirements

BACnet4MbedOS [9] uses a modified version of a BACnet stack originally created and provided as an open-source project on Sourceforge [5] by S. Kargs. Since the BACnet stack in itself only supplies BACnet communication abilities, all other necessary resources and interfaces are to be procured by the mbed platform. Apart from inevitable resource requirements such as memory and stack space, an interface linking the configured physical network is mandatory (see IV.B). This includes network specific protocols and stacks. Apart from the capability to support protocols required by the actual network and UDP datagrams, further protocol support is not mandatory for a BACnet/IP implementation. Other, not strictly required prerequisites, is for example the division of application tasks into separate threads. BACnet4MbedOS utilizes a single thread; executing specific, reoccurring BACnet stack tasks (see IV.I).

B. Ethernet IP network connection and UDP communication

To employ the flexibility and configurability of Mbed OS, the BACnet stack uses the provided *EthernetInterface* C++ API as shown in Fig. 4. By default, this API provides the necessary components to connect to an Ethernet based IP network. This includes the support of further protocols, such as DHCP (Dynamic Host Configuration Protocol). Due to the integration of IP stacks into Mbed OS, additional adaptions or protocol stacks are unneeded. For use with the *EthernetInterface* API Mbed OS provides a modified LWIP stack and additionally for a BACnet/IP communication over UDP, a UDP Socket API. A Datagram Socket is the center point of any BACnet/IP communication.

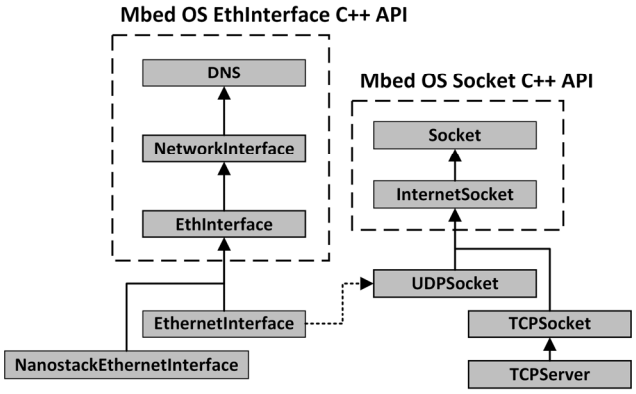

Fig. 4. Mbed OS Ethernet and Socket API

C. Device projection on BACnet objects

A device representation in a BACnet environment is the publishing of individual BACnet objects, representing data points. A unique ID, valid for each group of object types, identifies each of these objects (see II.C). BACnet4MbedOS provides an enumeration to validate the uniqueness of each ID and its actual assignment to an object and vice versa. As an example, the representation of a differential pressure sensor with analog voltage output is given. The sensor outputs a voltage from 0Volts to 10Volts, in dependence of a differential pressure of -25Pascal to +25Pascal. Two BACnet objects, an *AnalogInput* object and an *AnalogValue* object, represent this sensor in a BACnet environment. The analog sensor voltage is an information measured by an ADC (Analog to Digital Converter). Therefore, represented by an *AnalogInput* object. The pressure is a calculated value; hence, its BACnet object is an *AnalogValue* object. TABLE I. lists all mandatory object properties of an *AnalogInput* object according to the BACnet protocol. Other mandatory BACnet objects are comparable in terms or required property implementations.

TABLE I. Mandatory AnalogInput object properties

Analog Object Property	**AnalogInput property information**		
	Example	**Info**	**Add. Info**
Object Identifier	1	Unsigned integer	mandatory to all objects
ObjectName	AI_Input_01	Character string	mandatory to all objects
ObjectType	0	unsigned integer	mandatory to all objects
PresentValue	72.9	real	actual value
Status Flags	In_Alarm Fault	flags	
EventState	Normal	unsigned enum	
OutOfService	False	Boolean	
Units	Ampere	unsigned enum	

D. Change of Value subscriptions

Opposed to polling an object's *PresentValue* property, BACnet supports event-based notifications [3]. A client may send a *Subscribe COV* (Change of Value) service request, requesting a notification regarding changes of an object's *PresentValue* property. This applies to changes exceeding a set threshold value. For *PresentValue* properties, this threshold is defined as the *COV Increment*. TABLE II. lists different threshold rules and levels for a selection of standard BACnet objects. Alternatively, a notification is sent, if a change of the *Status Flags* property occurs. A BACnet device may respond to a COV subscription with either unconfirmed or confirmed COV notifications, allowing for the detection of offline subscribers after expiring a set response time.

TABLE II. BACNET COV THRESHOLDS AND LEVELS

BACnet Object Type	Notification Event	
	property	Threshold rule / level
Analog	PresentValue	change ≥ COV Increment
	Status Flags	any change
MultiState	PresentValue	any change
	Status Flags	any change
Loop	PresentValue	change ≥ COV Increment
	Status Flags	any change

E. Deployment and default configuration

BACnet4MbedOS is provided [9] as an open-source Mbed OS library, usable as a subproject during application development, e.g. with *mbed-cli* (see III.B). Part of the library is a *mbed_lib.json* configuration file. The JSON file provides project configurations and definitions. These scripts allow further definition of target or platform specific configurations and overrides. Generated defines and macros are collected in the *mbed_config.h* header file. This header is a collection of configuration parameters across all libraries utilized within the current Mbed OS project. Apart from vendor and application version information, BACnet datatypes supported by the stack during *writeProperty* service requests or the BACnet layer (see II.B) can be configured using JSON script files.

F. Proprietary properties

BACnet supports the definition of proprietary properties for objects. BACnet4MbedOS uses this to configure a devices IP configuration, such as the Netmask, via the *deviceObject*. The standard defines reserved property IDs within a range of 0 to 511. IDs > 511 may be used for proprietary IDs as described in the ASHRAE [4] standard by Clause 23. To define proprietary properties, BACnet4MbedOS defines valid property IDs within a unique enumeration list. Objects hold arrays, one for each group of required, optional and proprietary properties. Proprietary properties need to be added to an object's handler function, e.g. write property service request handler. Otherwise, confirmed request handlers treat requests made to unknown properties as erroneous and issue an error or reject answer.

G. BACnet object instantiation

After deployment, the device needs to instantiate a number of objects to describe its functionality via BACnet. The *deviceObject* as shown in Fig. 5 is unique and mandatory for any BACnet stack. Therefore, its instantiation is organized as a singular structure object. Opposed to that, other objects, e.g. *AnalogOutput* objects are limited only by code or hardware limitations. Due to this, general-purpose objects of the same type, are instantiated as an array of structure elements.

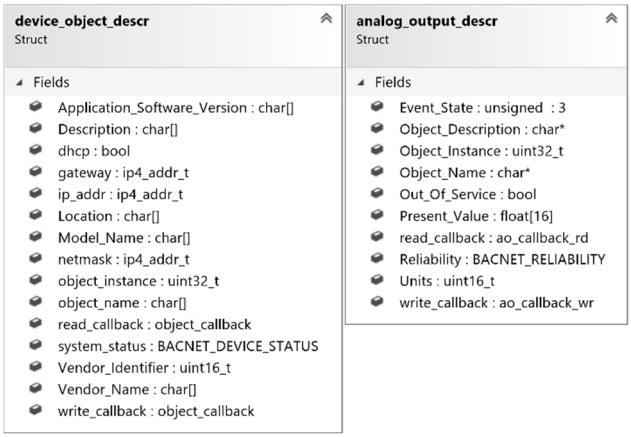

Fig. 5. BACnet DeviceObject structure

H. Object read/write callbacks

For the support of read and write property capabilities linking to resources outside the stack, objects can hold optional callback function pointers (see Fig. 5). These are executed by service request associated handler functions. Therefore, when an object property is accessed via service request and the object holding the property defines such a callback function. An example for such behavior would be a function utilizing an EventQueue (see IV.I) to defer the actual property access to a different thread. In general, these callbacks represent an option of BACnet4MbedOS to extend a service request handler's capabilities on a per object basis. This way granting application specific, object individual handler functionalities unrelated to the actual object type.

I. Task separation and deferral

To achieve low response times for BACnet interactions, a dedicated thread handles the processing of the BACnet stack. The thread continuously polls the Ethernet receive buffer, via Mbed OS API (see IV.B). If an APDU (Application Protocol Data Unit) packet containing a service request is received, a callback is referred to the *EventQueue*, as described in III.C. The queue then executes the actual deferral.

J. Property access from application code

Changes to properties of BACnet objects are possible from two different directions. On one hand, access to properties via BACnet is established via *read-* and *writeProperty* requests via the stack itself (see II.D). On the other hand, the application code accesses properties because of demands of the connected physical process. For each property type i.e. object type, individual get and set functions exist, to pay respect to

individual needs of e.g. datatype formats (analog vs. binary) or value limit restrictions enforced on individual properties.

K. BACnet4Mbed OS Performance

To evaluate the performance of the BACnet4MbedOS implementation, a STMicroelectronics STM3F746ZG microcontroller with Microchip LAN9303 switch chip running MbedOS v5.9.4 was tested. It has to handle multiple read property requests. A Windows PC issued the requests, running BACnet4Simulink [10] with MathWorks® MATLAB® R2018b. The model polled five *AnalogInput* objects at a rate of 20ms. The average response time across all five objects at a sum of 6000 requests per object is at ~7ms. Other tests with command line applications, implemented in C/C++, at the same Windows PC, resulted in notably lower response times, ~1.7ms average response time with a test run of 10.000 consecutive read property requests. The minimum and maximum response time for this test were 1.0ms and 17.4ms respectively. In parallel, the handling time of the BACnet4MbedOS application from receiving an NPDU (Network Layer Protocol Data Unit) up to sending the acknowledgement NPDU was measured. The test uses an oscilloscope to measure the duty cycle of a pin going high on receiving a NPDU and going low after transmitting the answer NPDU. The measured execution time is ~172µs (see Fig. 6).

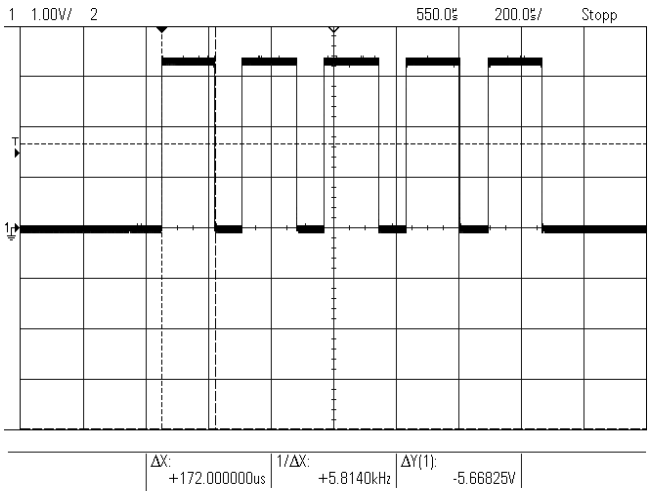

Fig. 6. BACnet stack NPDU execution time on controller

V. Conclusion

BACnet4MbedOS is the first BACnet stack dedicated to Mbed OS. It enables ARM Cortex-M microcontrollers to utilize a standardized protocol in IP based Ethernet environments. It simplifies the development of sensors and actors connected to automation networks. Representing an adaptable and configurable interface, BAcnet4MbedOS serves as a basis for the development of IoT devices in device-to-device communication networks. This includes implementations such as sensors, actors or servers of any kind in general.

References

[1] Bushby, S.T., August 1993. "Back to the Basics About BACnet" NIST internal document, reprinted by Trane as BAS-S-28

[2] Bushby, S.T., 1997. "BACnet: a standard communication infrastructure for intelligent buildings" Automation in Construction 6 (1997). Elsevier, pp. 529-540

[3] Bushby, S.T., Newman, H. M., Applebaum, M. A., November 1999. "NISTIR 6392, GSA Guide to Specifying Interoperable Building Automation and Control Systems Using ANSI/ASHRAE Standard 135-1995, BACnet" National Institute of Standards and Technology

[4] ANSI/ASHRAE Standard 135-2012, BACnet - A Data Communication Protocol for Building Automation and Control Networks, online, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. and ANSI Std.

[5] Kargs, S. BACnet Protocol Stack, January 2005, Available at: http://bacnet.sourceforge.net [Accessed 12/2018]

[6] ARM Ltd., Mbed OS, Feburary 2013, Available at: https://github.com/ARMmbed/mbed-os [Accessed 12/2018]

[7] ARM Ltd., Mbed OS LWIP Stack, September 2016, Avaiable at: https://github.com/ARMmbed/lwip [Accessed 12/2018]

[8] Goldschmidt S., Ziegelmeier D., LWIP Stack, October 2002, Available at: http://savannah.nongnu.org/projects/lwip/

[9] University of Wismar, BACnet4MbedOS, December 2018, Available at: https://github.com/ATM-HSW/BACnet4MbedOS [Accessed 12/2018]

[10] University of Wismar, Simulink4Bacnet, December 2018, Available at: https://github.com/ATM-HSW/BACnet4Simulink [Accessed 02/2019]

[11] Marquardt M, Bock C., MpBus-BACnet Gateway für den Einsatz in der Reinraumtechnik, September 2017