Designing a Development Board for Research on IoT Applications in Building Automation Systems

M.H. Schraven[a], C. Guarnieri[b], M.A. Baranski[a], D. Müller[a] and A. Monti[b]

[a]Institute for Energy Efficient Buildings and Indoor Climate, Germany
[b]Institute for Automation of Complex Power Systems, Germany
E-mail: mschraven@eonerc.rwth-aachen.de, cguarnieri@eonerc.rwth-aachen.de

Abstract

Recent advances in the development of Internet-of-Things (IoT) devices have enabled researchers to apply such on building automation systems (BAS). Especially in existing BAS, one big challenge is the diversity of communication methods. In such systems, gateways are required to connect various devices. However, while there are very few IoT-gateways able to interface a wide variety of common sensors and actuators, those often implicate high costs, entailing inacceptable investments for larger field tests.

To overcome this issue, we prototyped a low-cost plug-and-play and freely programmable IoT-gateway, which supports common analog signal interfaces like 0-10 V or 0-20 mA current loop, and digital ones via RS-485 serial communication. The gateway is based on the ESP32-PICO-KIT development board, a fully functional board with a microcontroller and embedded WiFi. Accordingly, we enhanced this board by adding custom-designed peripherals and developing the required firmware for acquiring sensor data and driving actuators.

In order to validate the functionality of the interfaces, we conducted an experimental test series. The experiments comprise measurements of inputs and outputs for either the IoT-gateway or the connected sensors and actuators. The results show an average relative error for analog interfaces of 7 %, hence being sufficient for building automation applications. The RS-485 was successfully tested with a Modbus RTU slave device.

Therefore, the prototyped IoT-gateway is directly applicable to both analog and Modbus-based sensors and actuators, shows acceptable errors in analog readings and can be manufactured at a relatively low price, facilitating test benches containing several plug-and-play gateways.

Keywords –

IoT, Gateway, Building Automation, Wireless Communication, ESP32

1 Introduction

In Germany, potential energy savings due to building automation system (BAS) improvements estimate roughly 20 % [1]. Additionally, in existing non-residential buildings, wireless IoT devices constitute a promising resource to enable BAS without causing high expenses due to elaborate cable installations. However, the beneficial installation of radio-based devices only applies if the configuration effort is not as high as in conventional systems on the other hand. Furthermore, local wired automation systems are well-known and widely adopted due to their reliability. However, there is no comprehensive study on a real BAS yet, which is completely operated via IoT devices. Hence, our research is focussed on the stability and properties of such systems utilizing several IoT devices.

When investigating IoT applications in BAS, the most cost-efficient approach is to use an existing infrastructure. Instead of exchanging sensors and actuators, gateways can be used. With BAS often comprising sensors and actuators with various communication interfaces, there are only few IoT-gateways directly applicable to all these devices. Some of the most popular communication interfaces used are 0-10 V and 0-20 mA analog signal transmission and bus-based communication e.g. via Ethernet, BACnet, LON, KNX and Modbus [2]. Some examples for multi-communication gateways are the BASremote at a price of 350 € [3], the EWIO-9180-M at 425 € [4], the UCM-316 at 325 € [5] and the WISE-4470-S250 at 400 € [6]. All these examples are rather edge controllers than decentral plug-and-play IoT-gateways. These edge controllers imply high costs when used for each sensor and actuator individually or limit one to research activities on decentralized summarized data processing. By contrast, already available open-source platforms like Controllino, PiXtend and UniPi Neuron, just to name a few, prove to be still expensive or not versatile enough for single transducer interfacing.

We thus identified the need to design an IoT-gateway that features only the necessary characteristics to enable IoT on single devices without causing inacceptable costs.

The rest of this paper is structured as follows: Chapter 2 starts with the deduction of requirements for the IoT-gateway, followed by a detailed description of the electronic design as well as the software implementation. In chapter 3, the conducted test series is described; chapter 4 summarizes the validation results and discusses the purchase cost. Finally, chapter 5 concludes with a summary and future possible improvements.

2 System requirements and design

From our experience with designing and operating BAS, we define the following properties for the IoT-gateway as mandatory requirements to allow for an accurate operation of BAS. These properties were mainly derived by the current technical building equipment, which was planned and used by the Institute for Energy Efficient Buildings and Indoor Climate for their new test hall.

- Timescale – Due to fluid and thermal inertia, but also delay of moving actuators, we suggest a physical and interpretation delay of maximum 1 s.
- Storage – Since the gateway will be connected to a wireless or local area network with constant access to online database storage systems, a limited internal storage capacity is requested for buffering purposes.
- Interfaces – For a start, we focus on analog signals and serial communication and thus omit the Ethernet-based communication. The gateway shall at least support 0-10 V signals, 0-20 mA signals and communication over RS-485.
- Resolution – Actuators such as valve actuators, volumetric flow controllers or pumps commonly receive their values in percent. Therefore, the analog output should at least resolve integer values ranging from 0-100, which represents a resolution of at least 7 bit ($2^7 = 128$ discrete values). Analog inputs are often used to receive signals from temperature sensors like resistance temperature detectors (RTDs) and negative temperature coefficient (NTC) thermistors. Assuming a range of 140 K, the gateway should at least read for one decimal, resulting in a minimum resolution of 11 bit ($2^{11} = 2048$ discrete values).
- Network connection – The gateway requires a way to connect to a local area network or the internet, e.g. via WiFi.
- Time tagging – For some control applications, it is necessary to tag each data measurement with a synchronized time. Hence, the IoT-gateway should offer a chronometric and time-reading possibility.
- Software development – In order to test different configuration setups, local or agent-based control strategies, remote service and maintenance concepts as well as different security and encryption functionalities, the gateway needs to be freely programmable. As Python is the most frequently used programming language at our institute, we require Python language support for our gateway.
- Power supply – The gateway should connect to existing sensors and actuators that are often supplied with industrial voltage level of 24 Vdc.

We use the ESP32-PICO-KIT [7] as a base for our IoT-gateway. The ESP32-PICO-KIT is a relatively cheap system on a chip by Espressif, yet providing solutions to several of the defined requirements. A detailed overview is presented in Table 1. ESP32-PICO-KIT properties

Table 1. ESP32-PICO-KIT properties [7]

Feature/requirement	ESP32-PICO-KIT
Timescale (physical, interpretation delay)	0.1 ~ 1 s
Storage	520 KB SRAM
Interfaces/resolution	12 bit ADC (0-1.1 V), 8 bit DAC (0-3.3 V), UART TTL (serial)
Network	WLAN/Bluetooth
Time tagging	RTC (internal real time clock) + NTP (network time protocol)
Software	Micropython support
Power	5 Vdc, 3.3 Vdc

2.1 Electronic Design

From Table 1, we can derive the required adjustments:

- Power supply voltage level,
- 0-10 V reading and writing voltage level,
- 0-20 mA current to voltage and voltage to current conversion and
- UART to RS-485 conversion.

2.1.1 Power Conversions

The IoT-gateway will be connected to 24 Vdc. As the ESP32 runs on a nominal power of 5 Vdc, we use a low dropout voltage regulator (LDO) to provide the requested conversion while being able to supply a current up to 1.5 A. Due to the 5 V operation, the ESP32 cannot provide a voltage output up to 10 V. Therefore, we also use a 10 V voltage reference (VREF), which serves as the upper reference for the 0-10 V output loop. The sensors/actuators and the IoT-gateway share the same ground.

2.1.2 0-10 V Input

As shown in Table 1, the ESP32 has a 12 bit ADC with an input voltage range of 0-1.1 V. This range has to be mapped to a sensor output ranging from 0-10 V.

The easiest and most cost-efficient solution to reduce voltage to a specific range is a voltage divider circuit (see Figure 1). This circuit converts the input signal according to Equation (1). Since the ESP32 ADC input impedance is not provided by the manufacturer, an operational amplifier (OPA) in buffer configuration is used as precaution to decouple the voltage divider from the ADC, thus avoiding measurement errors due to load effects.

$$V_{ADC} = V_{in} \cdot \frac{R_{13}}{R_{14} + R_{13}} \qquad (1)$$

As the ESP32 is a low-cost device, its ADC is not of the best quality. It is only assumed to be linear in a range of 100 mV to around 950 mV [8]. Therefore, we selected the resistors so that input voltages of 0-10 V match ADC input voltages of 0-0.91 V. Note, that this reduces the effective resolution of originally 12 bit to about 3400 discrete values, which is still considerably higher than the required 11 bit resolution.

2.1.3 0-20 mA Input

The 0-20 mA input current loop requires a conversion from current to voltage so that the ADC can read its value. The easiest solution is to use a single resistor providing a voltage following Ohm's law (see Equation (2)). Again, we add a buffer to decouple the load resistor R_3 from the ADC and select the resistor so that the ADC input voltages match 0-0.91 V for input currents of 0-20 mA.

$$V_{ADC} = I_{in} \cdot R_3 \qquad (2)$$

2.1.4 0-10 V Output

The DAC of the ESP32 has a maximum output voltage of 3.3 V. In order to achieve higher output voltages, this voltage has to be amplified. For this purpose, we use a rail-to-rail OPA in non-inverting amplifier configuration, as shown in Figure 2. By adapting the resistor values, the gain, and hence the output voltage is adjustable, as long as it is below the power supply. The OPA drains a current of just 160 µA, allowing to be powered with a high precision voltage reference (see Section 2.1.1). Whilst the internal voltage supply of the ESP32 is 3.3 V, the DAC may output maximum voltages of 3.2 V. Adopting a conservative approach, we select the resistors to map the DAC output voltage of 0-3.15 V to an output voltage of 0-10.1 V. Because this output is used to control actuators, the output voltage must exceed 10 V, for instance to ensure that the actuator is actually capable of fully closing a valve. The resistor values can be derived by Equation (3). Regarding the effects on the final resolution, the safety margin causes a negligible loss of 12 discrete values.

$$V_{out} = V_{DAC} \cdot \left(1 + \frac{R_{11}}{R_{10}}\right) \qquad (3)$$

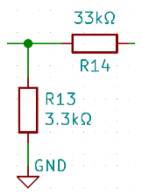

Figure 1. Voltage divider circuit of our IoT-gateway, transfers 0-10 V to 0-0.91 V

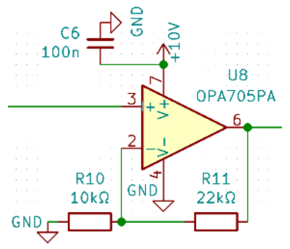

Figure 2. Voltage amplifier circuit of our IoT-gateway, amplifies 0-3.15 V to 0-10.1 V

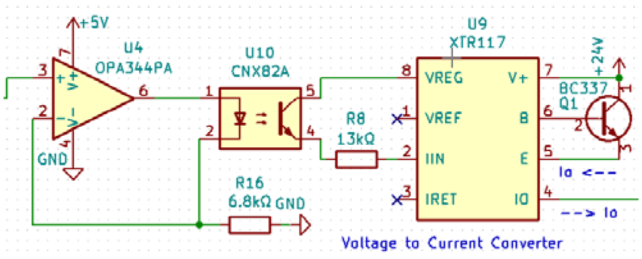

Figure 3. Current loop driver of our IoT-gateway, converts 0-3.15 V to 0-21 mA

2.1.5 0-20 mA Output

We realize voltage to current conversion by resorting to the XTR117 by Texas Instruments, a commercial precision current driver that can be configured both in true and live zero configurations at a very low cost. Since both the DAC and the XTR117 share the same ground, the driving block is isolated from the control block by means of an optocoupler. However, since the relation between the current flowing in the diode and photo-generated current in the bipolar junction transistor is non-linear, the linear control on the former does not result in an equally linear control of the latter. The circuit illustrated in Figure 3 follows the characteristic described in Equation (4), which we derived by measuring the driver current for different values of control voltage.

$$I_{out} = \mathrm{a} \cdot V_{DAC}{}^{b} \tag{4}$$

2.1.6 UART TTL to RS-485 Conversion

The Universal Asynchronous Receiver Transmitter (UART) interface is an interface used to transfer serial data asynchronously. The asynchronous communication does not require a clock signal but rather expects a start and an end of a transferred message [9]. On the ESP32, the UART runs on 5 V. A further difference is the line driver, which converts single ended UART signal to a bi-directional differential signal resulting in two data lines, Data A and Data B. The signals for RS-485 usually range between +/- 1.5 V [9]. For the signal conversion, we use a commercially available UART to RS-485 converter, which also features automatic flow control. Without automatic flow control, a digital control signal is required, in order to assign one line the for communication, as it is not possible to transfer data over both reading and transmitting line at the same time. This, in particular, complicates timing the communication, hence we decided to use a module with automatic flow control.

2.1.7 Reference Voltage Calibration

The ADC and DAC lack quality in terms of linearity that are mainly due to noise and varying reference voltages between different microcontrollers. Hence, we perform an automatic ADC and DAC calibration by calculating two points used for linear interpolation.

A reference voltage for the ESP32 is set, emitted and redirected to two ADC pins, after the voltage is reduced to 1/3 and 2/3 of the reference voltage, respectively. With this construction, the ADC is calibrated. Afterwards, the DAC redirects its output to an ADC pin, reading the provided voltage from the DAC.

2.1.8 Low-Pass Filter

Input and output stages are low pass filtered using first order RC filters to reduce the noise. The adopted bandwidth of 15 Hz is selected to allow all signals in timescales down to 1 s without significant delays and attenuations.

2.1.9 Summary

The ESP32-PICO-KIT is expanded by adding a 24 Vdc power conversion, 5 circuits for each of the individual communication interfaces and a calibration circuit. Figure 4 depicts a view of our finished prototype. The schematic and Printed Circuit Board (PCB) layout can be examined on the publicly available GitHub repository [10].

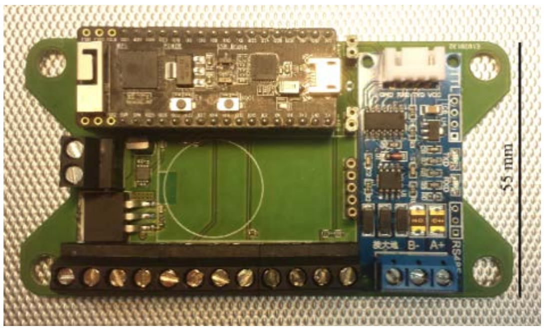

Figure 4. Assembled IoT-gateway PCB prototype from our defined requirements

2.2 Software Design

We programmed the firmware in Python code. However, the microcontroller could also be programmed with other languages like Arduino or C++. The storage space on the ESP32 is limited, therefore we use Micropython, which is a lean Python implementation and was especially developed for microcontrollers [11]. Because the basic Micropython firmware does not allow for setting a reference voltage, we used the LoBo Micropython implementation, that is freely available on GitHub [12]. The GitHub page also contains some instructions on how to flash the firmware onto the ESP32 (we use the esptool for flashing the firmware and ampy to transfer files over serial connection). Following the instructions, the ESP32 accommodates a file system, so the software can be written with any Python programming environment or plain text editor and be transferred onto the ESP32.

When booting the ESP32 with Micropython firmware, two files, boot.py and main.py, are always executed one after the other. The following sections explain the individual parts of the software in more detail.

2.2.1 Main.py

In this file, an instance of the board class is created. All functions, which are used for addressing the analog signal transmissions and Modbus communication, are implemented within the board class. The functions may be called from the instance within a Python shell that is available on the ESP32 when e.g. connecting via serial connection.

2.2.2 Board.py

The board.py is the main file for interfacing the different communication interfaces and provides the corresponding functions. In this file, a class “board” is defined. All parameters are stored within the class object. The following methods are available:

- __init__(self): This function is called when creating an instance of the board class. Parameters are fetched from the parameters.py and assigned to the attributes. In addition, the ADC and DAC pins for 0-10 V input and output as well as 0-20 mA input and output are assigned. After this, a Modbus instance is created. Finally, the methods for setting a reference voltage and executing the calibration are called.

- set_ESP_referenceVoltage(self): This function sets the reference voltage to the predefined pin (in our case IO27, see schematic [10]).
- calibrate(self): This function calls the ADC and DAC calibration.
- calibrate_adc(self): In this function, the ADC regression parameters are calculated and slope and intercept are returned.
- calibrate_dac(self): In this function, the ADC regression parameters are calculated and slope and intercept are returned.
- Four methods for either ADC or DAC conversions between digital and voltage values: This allows an incremental and consistent calculation for the reading and writing functionalities, but also requires to define the ADC and DAC characteristic at least once.
- Four methods for either reading or writing analog voltages and currents: The reading and writing methods contain the relations described in chapter 2.1.
- Modbus-related functions are callable by the Modbus instance.

2.2.3 Parameters.py

This file contains all predefined parameters. E.g.:

- Pin numbers
- Predefined digital and voltage values for calibration
- Parameters of the relations described in chapter 2.1.
- Modbus-related parameters: Definition of the physical serial port and other specific parameters like the baudrate of the connected device, the Modbus slave address and register lengths.

2.2.4 Modbus

For Modbus, the modules uModbusSerial.py, uModbusFunctions.py and uModbusConst.py are available. These files are slightly adjusted versions of the uModbus package for Micropython [13]. The uModbusSerial.py being the main file provides a class with standard reading and writing functionalities for registers and coils. Our changes mostly concern reading the UART interface.

2.2.5 Summary

The software implementation features a board class that provides the interfacing methods for reading and writing analog voltages and currents as well as receiving and sending Modbus signals. At current state, no automatic sensing or control and logging routines are implemented. When booting the ESP32, a “board” instance is created. This board instance is used via console from serial connection. Modbus-specific functions were taken from the Micropython uModbus module. Parameters may be accessed and changed in the parameters.py module. The complete software implementation is available in the GitHub repository [10].

3 Test series

In order to validate the functionality of the interfaces, a test series is conducted. For the validation of analog reading and writing, we use a test setup consisting of a Programmable Logic Controller (PLC) and terminals for 0-10 V input and output as well as 0-20 mA input and output. With these terminals, it is possible to provide constant voltages and currents or read those generated by the IoT-gateway. We validate the RS-485 interface with a Modbus RTU slave device – an electric valve actuator [14]. The Modbus protocol works with digital values stored in registers and coils; a description of the Modbus registers is provided by the manufacturer [15]. Table 2 provides an overview of the used testing hardware components.

Table 2. Test series hardware

Tested gateway interface	Testing device
-	CX5130 (PLC)
0-10 V input	EL4008
0-10 V output	EL3008
0-20 mA input	EL4018
0-20 mA output	EL3048
RS-485	LM24A-MOD

3.1 Setup

We investigate the analog reading and writing functionality by providing predefined voltages and currents with either the PLC terminal or the IoT-gateway and measuring the corresponding value on the other side. For 0-10 V, we therefore varied the voltage with a step width of 0.5 V, for 0-20 mA we used 1.0 mA steps. Accordingly, we compared the IoT-gateway’s signal to the terminal’s signal. Because of noise, the signal may vary at equal conditions. Therefore, we read 1000 samples and calculate the average value.

The Modbus communication is validated by simply writing a set point to a register, waiting for the actuator to move and reading its corresponding actual value afterwards. We wrote on register 1 a value of 2500 and 7500 and read the related value on register 5. Register 1 is related to the positional set point, which ranges from 0-10000, where 10000 corresponds to 100 %. Register 5 reads the actual position, again ranging from 0-10000.

3.2 Calibration issues

With the proposed calibration method, the ADC and DAC results show severe deviations from expected. However, the problems with noise and accuracy due to different reference voltages are known issues to the manufacturer of the ESP32, Espressif, that provide in their own software development kit and programming guide some useful calibration functions that depend on the internal reference voltage.

For chips produced after the beginning of 2018, the internal reference voltage is measured and directly fused to each chip and hence the calibration can be automated easily. Since the chips of our ESP32-PICO-KITs were produced in 2017, their reference voltage cannot be read directly from the chip. Therefore, in order to use the calibration functions provided by Espressif, we measured the reference voltage and calculated slope and intercept for the linear regression. The calibration functions by Espressif lead to results that are more accurate; consequently, we use their calibration method over our two-point regression. The next section gives an overview of the validation results with this calibration applied.

4 Results and Discussion

The validation results for 0-10 V input and output are summarized in Figure 5, Figure 6 illustrates the corresponding results for 0-20 mA input and output. Table 3 summarizes the maximum and average deviations for the analog communication.

Even with the calibration functions given by Espressif, the results show significant errors for both small voltage and current values. On the one hand, the internal ADC characteristic typically exhibits a negative offset of almost 75 mV that results in the impossibility of reading any value below that threshold, i.e. approximately 750 mV and 1.5 mA for the voltage input interface and the current one respectively. On the other hand, the DAC characteristic shows an offset of almost 300 mV, thus resulting also in this case in a reduced effective range.

With regard to control purposes, the DAC should always be able to apply a zero voltage; besides that, the DAC deviations are probably negligible, since, in many applications, control algorithms rather rely on differences between set and actual values to produce a specific output for an actuator, for instance in case of simple proportional–integral–derivative (PID) control. On the contrary, the ADC readings should be accurate and hence improved on the whole range. This, for instance, applies in case of cascaded PID control where different single sensor errors could add up to significant gain errors. However, the minimum range limitation rather affects actuator position feedback, as e.g. temperature or humidity sensors normally operate far from the minimum of the range. Besides that, following the example of simple PID control, the sensor data acquisition inaccuracy would lead to a different gain, but would not change the general behavior. In order to fully evaluate the limitations for BAS operation, a detailed study is required addressing specific criteria like cyclic communication times, packet size, delays et cetera to guarantee the system's controllability.

At this stage, excluding the error below the 20 % of the input/output range, maximum absolute errors are 0.08 V and 0.16 mA, respectively, which shows that the device is usable for operating with standard voltages 2-10 V and standard currents 4-20 mA for control purposes.

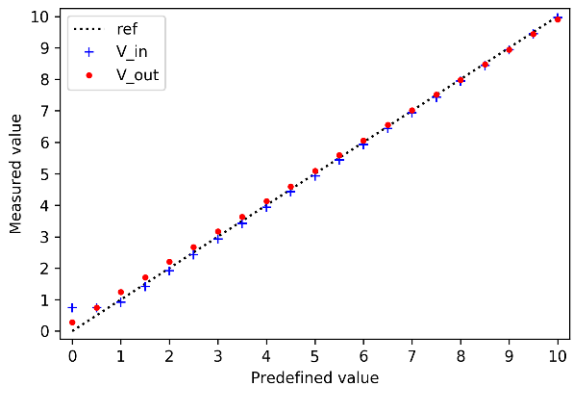

Figure 5. Validation of 0-10 V input and output

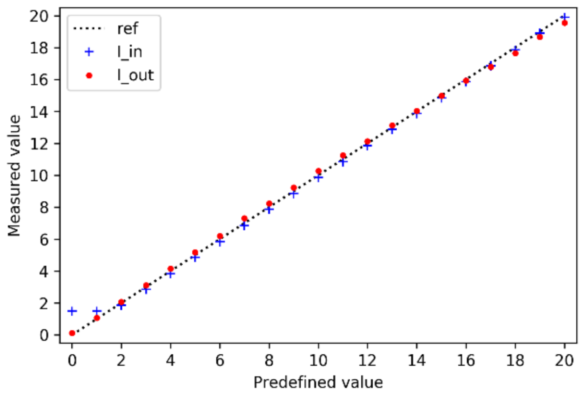

Figure 6. Validation of 0-20 mA input and output

Table 3. Analog relative (e_r) and absolute (e_a) maximum, minimum and average errors

Interface	0-10|2-10 V in	0-10|2-10 V out	0-20|4-20 mA in	0-20|4-20 mA out
	[%]	[%]	[%]	[%]
$e_{r,max}$	49.5|4.19	48.6|10.1	50.0|3.88	7.64|4.32
$e_{r,min}$	0.42|0.42	0.36|0.36	0.53|0.53	0.77|0.77
$e_{r,avg}$	4.35|1.45	6.39|2.43	4.29|1.44	2.76|2.32
	[V]	[V]	[mA]	[mA]

$e_{a,max}$	0.75|0.08	0.29|0.20	1.5|0.16	0.46|0.46
$e_{a,min}$	0.04|0.04	0.03|0.03	0.11|0.11	0.08|0.12
$e_{a,avg}$	0.11|0.07	0.12|0.09	0.22|0.14	0.21|0.24

For the Modbus device set point of 2500, we read a value of 2501, for 7500 we read 7502 on register 5. Modbus is a digital communication, hence not causing any deviations. The very small deviations may occur due to the accuracy of the actuator position.

From the validation results, we conclude following assertions:

- Due to varying internal voltages between different ESP32 modules, the proposed calibration method did not show the desired improvement on the ADC and DAC accuracy. A different method should be proposed in the future to calibrate the conversions between analog and digital values automatically.
- Within the range of 2-10 V and 4-20 mA, the IoT-gateway was able to read analog voltages and currents at an average error of 1.45 % and 1.44 %, respectively. The maximum deviation amounted to 4.19 %. In the same range, writing of analog voltages and currents accounted for average errors of 2.43 % and 2.32 %, respectively. The maximum deviations for writing amounted to 10.1 %. For smaller voltages and currents, the ADC is not able to dissolve them. This issue should be addressed in the future, since reading the actual value accurately is important when calculating a specific output. The writing functionality yielded an acceptable accuracy.
- The PLC terminal EL3048 has an input impedance of about 85 Ω. The current output loop was tested with different loads and showed that a maximum load of 250 Ω can be handled. Further research on input impedances of BAS devices is necessary to assure a general suitability for BAS applications.
- Communication via RS-485 was tested with a Modbus device. The set point register was written and the actual value register read with success. Very small deviations occurred, probably due to the electric motor accuracy.
- The IoT-gateway transmitted signals within 1 s for all tested interfaces.

The technical requirements are met. However, one question remains: How much does this multi-gateway cost compared to the commercially available devices?

4.1 Prices

With the assumption, that a larger field test would require around 50-100 IoT-gateways, we examine the prices for all used components. Figure 7 shows the cost distribution: except for the RS-485 converter, all component prices are fetched from www.mouser.de [16]. This diagram shows that the ESP32-PICO-KIT costing 8.73 € accounts for almost half of the total costs. The circuits for 0-10 V in- and output as well as 0-20 mA input are equally distributed with costs of about 1 € each, the RS-485 module falling little behind with 1.18 €. Because of the high precision reference voltage for the OPA705, the power supply costs 4.34 € and thus causes the second highest costs in total. Since the OPA705 is the only part in need of 10 V, this design should be revised in order to further reduce the costs. However, this component could also help utilizing and improving an automated ADC calibration. The current output loop yields the fourth highest costs and should be revised as well, in order to allow for devices with higher input impedances. The total component price is 23.57 €.

In addition to the components, the PCBs have to be produced and all parts have to be assembled. The costs for 100 PCBs including assembly and shipping account for roughly 400 € corresponding to 4 € per piece. In total, the purchasing costs for one IoT-gateway at quantities of 100 amount to about 28 €. From a vendor's perspective, this price could obviously be reduced by purchasing the components in larger amounts or entering special contracts with manufacturers. However, aside from researching aspects, a single-gateway would probably be more suitable for industrial BAS applications.

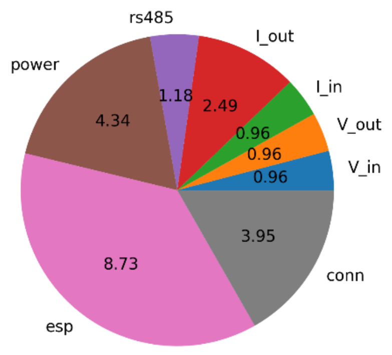

Figure 7. Cost distribution of the IoT-gateway; costs per gateway in € for 100 pieces ordered

5 Summary and Conclusion

In this paper, we designed an IoT-gateway for research on applications in BAS. From our experience, we derived technical requirements for an IoT-gateway in BAS applications. More specifically, we defined:

- Timescales for physical and interpretation delay
- Storage and resolution requirements
- Commonly used signal transmission interfaces
- Software requirements
- Network and time-tagging requirements

- Common industrial power supply in BAS

Utilizing the ESP32-PICO-KIT as a base module satisfies several of the defined requirements such as network connection and time-tagging abilities, ADC and DAC functionalities with appropriate resolution or enough storage capacity for buffering purposes. In order to provide the required interfaces, we designed the additionally necessary peripheral circuits accordingly to allow for analog as well as bus-based communication via RS-485. The ESP32 is a freely programmable WiFi controller; the software addressing all interfaces was written in Micropython, however, it could be programmed with different languages either.

To validate the designed circuits for the different communication interfaces, a test series was conducted comparing target values to measured values. In the validation, analog readings showed severe issues when reading very small values. The average relative errors were around 4.3 %. The analog writing functionality resulted in deviations that were higher than the deviations in reading. In BAS, control algorithms often use differences between set and actual values to calculate an output, hence we conclude that the reading functionality requires improvement. However, further investigations should aim for specifying the exact limitations for BAS operation. Besides that, the analog writing functionalities require small changes in order to at least ensure reaching both upper and lower limit. Additionally, the current output loop is only able to produce a current for devices with resistances up to 250 Ω. For RS-485, we successfully tested a Modbus device. This interface could also be utilized to realize BACnet communication. Further development should focus on extending the firmware and testing BACnet support. In general, the technical requirements could be met showing some restrictions.

Additional improvements include logging routines and streaming of measurement results over WiFi as well as the implementation of control loops.

As regards purchase costs, the total costs amount to roughly 28 €, hence being far below comparable IoT-gateways like a shielded Raspberry Pi with a retail price of 100 €, a Controllino at 200 € or edge controllers at 300-400 €.

6 Acknowledgement

EUROPEAN UNION
Investing in our Future
European Regional
Development Fund

We thank the ERDF for their financial support.

References

[1] Lonmark Deutschland e.V. Energieeffizienz automatisieren, Aachen, 2011.

[2] MeGA, Marktstudie, Marktstudie Gebäudeautomation, Schweiz, 2012.

[3] Control Consultants Inc. BASR-8M. On-line: https://controlconsultantsinc.com/basr8m-contemporary-controls-bas-remote-master-6-universal-io-2-relay.html, Accessed: 04/01/2019

[4] Arigo Software EWIO-9180-M. On-line: https://www.arigo-software.de/de/shop/110910.html, Accessed: 04/01/2019.

[5] Monotaro UCM-316. On-line: https://www.monotaro.com/p/0040/5912/, Accessed: 04/01/2019.

[6] EK3OT WISE-4470-S250. On-line: https://ekzot.com.ua/product/wise-4470-s250/, Accessed: 04/01/2019.

[7] Espressif ESP32-PICO-KIT Getting Started Guide. On-line: https://docs.espressif.com/projects/esp-idf/en/latest/get-started/get-started-pico-kit.html#, Accessed: 07/01/2019.

[8] Espressif ESP32 Datasheet. On-line: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, Accessed: 07/01/2019.

[9] FTDI Ltd. What is UART? On-line: https://www.ftdichip.com/Support/Documents/TechnicalNotes/TN_111%20What%20is%20UART.pdf, Accessed: 07.01.2019.

[10] RWTH-EBC AixOCAT. On-line: https://github.com/RWTH-EBC/IoT-Gateway/, Accessed: 31/01/2019.

[11] Damien George Micropython. On-line: https://micropython.org/, Accessed: 07/01/2019.

[12] Loboris Micropython for ESP32. On-line: https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo, Accessed: 04/01/2019.

[13] uModbus. On-line: https://github.com/pycom/pycom-modbus/tree/master/uModbus, Accessed: 04/01/2019.

[14] Belimo LM24A-MOD. On-line: https://www.belimo.eu/pdf/e/LM24A-MOD_datasheet_en-gb.pdf, Accessed: 04/01/2019.

[15] Belimo Modbus-Register Description. On-line: https://www.belimo.ch/pdf/e/AirWater_Modbus-Register_en.pdf, Accessed: 04/01/2019.

[16] Mouser Electronics Inc. https://www.mouser.de, Accessed: 30/01/2019.