Equally Distributed Bus-Communication Access Rights for Inter MCU Communication Using Multimaster SPI

Manuel Dentgen(✉), Sebastian Renner, and Jürgen Mottok

Laboratory for Safe and Secure Systems (LaS3), Technical University of Applied Sciences, Seybothstraße 2, 93053 Regensburg, Germany
{manuel.dentgen,sebastian1.renner,juergen.mottok}@oth-regensburg.de

Abstract. With the rising complexity and processing power of modern computer systems, the amount of MCU on a single PCB also rises. These microcontrollers often need to communicate with each other to exchange payload and control information in a bidirectional manner. Today's well-established communication protocols in MCUs either do not fit modern transmission speed requirements or do have an inappropriate master-slave attribute, which does not allow the communication partners to have equal bus access rights. Therefore, this paper introduces an extension of the Serial Peripheral Interface (SPI) to allow an equally distributed access right for the communication interface between two microcontrollers. It simultaneously does fit modern transmission speed requirements of a common network interface, so that the message transmission does not constitute a bottleneck in data processing. Besides the protocol design, we do also provide a first prototype implementation, which constitutes a proof of concept.

Keywords: Multimaster · SPI · Communication · Master · Slave · Bidirectional · Equally distributed transmission rights · Microcontroller · MCU · Embedded · Ethernet · Inter · Controller · Conversion · Flow control · Multimaster · Serial peripheral interface

1 Introduction

Within the context of the research project *Energy Safe and Secure System Module* (ES^{3}M) [9] at the *Technical University of Applied Sciences* (Ostbayerische Technische Hochschule - OTH) in Regensburg, a module for the separation of the security protocol layer from the network stack is developed. To achieve this, the individual tasks of those layers are divided among several Microcontroller Units (MCUs) on a single Printed Circuit Board (PCB). The goal of this concept is to get a higher security level, since the sensitive security mechanisms are outsourced to a distinct centralized controller, which is not directly accessible from the external network. At the same time the controller, which can be accessed from the external network, has no security tasks. A third controller

A. Brinkmann et al. (Eds.): ARCS 2020, LNCS 12155, pp. 200–212, 2020.
https://doi.org/10.1007/978-3-030-52794-5_15

enables the module to communicate with an additional independent and isolated communication interface. This principle is shown in Fig. 1.

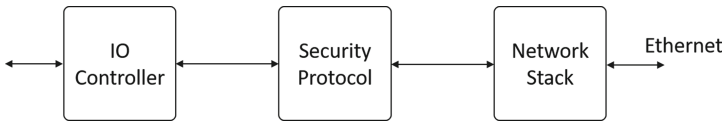

Fig. 1. Task separation of the security protocol layer and the network stack with the third Input/Output-controller on the left

1.1 Contribution

During the development of the research project it has become apparent, that the task separation principle used can easily be adopted to other application fields, e.g. for automotive or aviation. The third controller could communicate with an already existing sensor or actuator by connecting them via a standard communication interface. Our concept shall enable the conversion of low-level communication networks to Ethernet with an increased security level in those fields, without the need to exchange every single device, but by simply including this kind of module in a network.

However, the used MCUs only contain a single *Ethernet Media Access Control* (MAC) interface, which is used for the communication with the external network. This results in the necessity to implement another communication channel between the microcontrollers. The approach for this has to reach a similar transmission speed to what Ethernet already offers, to not create a bottleneck in data processing. It also has to allow bus access control on both sides with equal rights to allow an unconditional transmission in both directions. This paper will introduce such a communication interface.

1.2 Structure

The paper is structured as follows. In Sect. 2, some similar concepts to the one introduced in this paper are presented and compared to each other. It shall also be clarified why those are not fitting for our kind of task. Then in Sect. 3, the idea and theory behind the introduced architecture is presented, while considering the points previously mentioned. Section 4 focuses on the implementation of the concept within the context of the research project. Next, in Sect. 5 a proof of concept is presented, because the final implementation is not yet finished due to limitations of the current hardware. Section 6 summarizes the current state of the inter-controller communication and gives a brief outlook on future work.

2 Related Work

The idea of connecting several microcontroller units together and thereby generating a network of multiple controllers is not a new task in the embedded world. Various connection and design approaches have been made over the last years, all with the goal to achieve a fast, robust and secure communication between multiple controllers.

Niedermaier et al. [4] introduced a robust communication for the Industrial Internet of Things (IIoT) sector. They provide a concept to connect a communication interface with a following proof of concept. The presented architecture is similar to the one presented in this paper. However, they have used a standard serial peripheral interface for the inter-controller communication, with one of the controllers being the SPI master. This implies that the two communication partners are not equally permitted to access the communication bus. Besides other properties, this is not suitable for our project. We desire an inter-controller communication where both communication partners are fully authorized to access the bus.

Peng et al. [5] have presented a similar approach to our idea. They have realized an inter-controller communication using a dual-port Random Access Memory (RAM), and with that converting *RS232* or *RS485* to an Ethernet interface. However, they have developed a non-protected Ethernet endpoint, while we want to use a loop-through with an additional security mechanism.

Szekacs et al. [8] have implemented an inter-controller communication using the two interfaces SPI and Inter Integrated Circuit (I^2C) simultaneously. Their original goal was to connect a huge amount of sensors to one master using the I^2C-interface. As they have found out that they are not able to connect the estimated amount of sensors using only this communication type, they decided to extend their layout with several microcontrollers, all having their own connection to the sensors and communicating via SPI to the master. Maemunah et al. [3] have implemented a similar concept to them, by connecting multiple sensors to a MCU and sending all the measured data via a single interface to a processing master. The master is again the reason why we cannot use these approaches, since we want to use an equal bus access method.

The discussion shows that there are many inter-controller connection techniques, but none of them fitted perfectly for our task. Therefore, we have elaborated our requirements for the inter-controller communication more precisely and compared it to well established protocols with the goal to find a fitting interface.

3 Protocol Design

The previous chapter showed which elaborated communication systems are already available and why those are not suitable for the system presented here. Therefore, this chapter clarifies the requirements of the aspired communication interface. On the basis of this information, some well established communication protocols can be evaluated and eventually one is chosen.

3.1 Abstracting the Problem to Two Controllers

While speaking of three controllers for the research project prior to this section, the actual aspired communication interface represents a point-to-point (PTP) connection between two MCUs, which will be used twice originating from the centralized security controller. Therefore the problem can be abstracted to a data transmission between two controllers which is shown in Fig. 2. This represents another security benefit next to the task separation. The two connections are not implemented via a single bus system, but are separated from each other to prevent the data from being intentionally looped past the actual security controller. Thereby we achieve a higher security level, already during the hardware design phase.

Fig. 2. Communication principle broken down to two controllers

As mentioned before, an important setting for our communication protocol is the transmission speed between the controllers. The connection of the network controller can have a transmission speed of up to 100 Mbit/s. To not constitute a bottleneck in data processing, the inter-controller communication should also achieve a similar transmission speed to this extent. Another important property of the protocol is the fully equal transmission right in both directions. The reason for this is the possibility for both controllers to start a communication, as independent asynchronous data transmission is possible in our architecture. However, this results in the need for a full duplex transmission or the presence of a flow control between the two controllers, as no data shall be lost during transmission. These three aspects are the essential prerequisites for the transfer protocol.

3.2 Communication Protocols

With this information, some already well-established communication protocols can be elaborated and compared to each other. Communication protocols are sufficiently available, all having different advantages and disadvantages compared to each other. Examples for well-established communication interfaces, used by embedded devices in our daily life, are *UART*, I^2C, *SPI*, *CAN* or *Ethernet*. Besides those, there are a lot of proprietary communication protocols, which are specially adapted to an application by the developer of a device. Besides the aspect that it is difficult to get access to these proprietary interfaces, we have decided to use and extend a widespread protocol to make it available for a wider range of microcontrollers.

As already mentioned, Ethernet is not included in the evaluation due to the insufficient amount of hardware-modules on a single MCU. If it was possible to connect the controllers via several Ethernet networks, this would be the preferred solution for our problem. To compare the well-established interfaces, some characteristics of them have to be clarified. This includes transmission direction and the usage of a clock signal.

Transmission Variants. Transmission protocols can be subdivided according to their possible transmission directions in relation to the time course. There are two different types of transmission types which can be considered for our application. They are called *Half-Duplex* and *Full-Duplex*. Half-Duplex describes the possible data flow in both directions, but not at the same time. This means that a message transmission between controller 1 and controller 2 can only take place in one direction at a time, which would be a clear limitation in our system. Full-Duplex on the other hand allows transmission in both directions at the same time. To make this possible, the transmission is usually realized on two different channels (lines). UART and SPI offer a Full-Duplex transmission, while I^2C and CAN are Half-Duplex due to restrictions caused by fewer lines.

Clock Line. Next to the transmission direction, the protocols can also be divided into the groups *asynchronous* and *synchronous*. The difference here is the presence or absence of a clock line. This means that with synchronous protocols there is an additional line over which a clock signal is transmitted. It is always generated by one of the communication partners and read by the others. A valid data bit can then be generated and detected at the falling or rising edge of the clock line. This allows independent operation of the two controllers and simultaneously synchronous data transmission, because edge sampling is one way of resynchronizing the transmission. When using an asynchronous interface, a valid data bit has to be detected by a specified bit length. The critical part with asynchronous protocols is, that the controllers have to work synchronously during a transmission. When the internal processing clock of the controllers and with that the calculation of the bit length gets out of sync, a reliable detection of the bit states can no longer be guaranteed. This effect gets worse when transmitting large messages. It also results in the fact that asynchronous protocols usually do allow a lower maximum transfer rate than synchronous protocols. While UART and CAN are asynchronous protocols, SPI and I^2C belong to the category of synchronous protocols. A further detailed comparison of several serial communication protocols can also be found at [2]. We have summarized the attributes described in this chapter in Table 1, including the four presented communication protocols:

Next to the comparison of the already mentioned aspects, it is also possible to compare the energy consumption. Solheim et al. [6] have done exactly that with I^2C and SPI. They found out that I2C has a higher energy consumption than SPI and assume that the necessary pull-up resistors for the I^2C bus are the reason for this.

Table 1. Comparison of the attributes of different communication protocols

	Transmission direction	Clock line
SPI	Full-Duplex	synchronous
I^2C	Half-Duplex	synchronous
UART	Full-Duplex	asynchronous
CAN	Half-Duplex	asynchronous

3.3 Selected Communication Interface

The previous section shows clear advantages of SPI in comparison to the other interfaces presented. The higher transmission speed, the Full-Duplex attribute and the low energy consumption of SPI has led to the decision to use this interface in our system.

The well-established Serial Peripheral Interface was originally designed by Motorola Inc. to realize a fast, robust and synchronous data transmission between a single master and several slaves [7]. With this transmission protocol, the master always is the initiator of a message transfer, while a slave can only send an answer to a prior sent request. Furthermore, the master is generating the clock signal and with that the possibility of synchronizing the controllers. This concept fitted most implementations in the embedded world so far, because usually circuits had a single microcontroller which represented an intelligent unit, while all other integrated circuits have just been passive and non-intelligent (e.g. RaspberryPi[1], BeagleBoneBlack[2]). However, this method cannot fulfill our requirements of equally distributed communication rights between two microcontrollers. The one-sided initiation possibility always results in one of the controllers not being able to start a communication, which is not suitable for the in this paper described task. A consequence of this fact is the extension of the serial peripheral interface, while other attributes of the interface do already fit pretty good into our task.

Additionally Tongsan et al. [10] provide a software-defined inter-processor communication for embedded systems. They suggest to realize a software layer with a well-defined API to make the controlled hardware exchangeable. We have decided to use and integrate this technology into our system in order to remain flexible for possible extensions or changes as SPI still has its downsides.

4 Implementation of the Communication

After describing the theoretical background of the communication presented in this paper, we go on to the actual implementation. To do so, we briefly present aspects of the research project and clarify some necessary details of the specific controllers.

[1] https://www.raspberrypi.org/.
[2] https://beagleboard.org/black.

4.1 Context of the Research Project

The research project, for which this inter-controller communication was developed, essentially consists of the three microcontrollers, which do implement the task separation of security protocol and network stack. In the context of the project, the TLS protocol for the security mechanism with an underlying TCP connection is used. This is depicted in Fig. 3. The three controllers are of the type STM32H7[3], which provide many interfaces established in the embedded world, but, like already mentioned, just a single Ethernet interface. In theory, other MCUs would fit into our project, but we have chosen this one, as it has a high transmission speed Serial Peripheral Interface compared to other controllers. Nevertheless, future developments on other controllers should remain possible.

Like mentioned in the beginning of this paper, the main aspect of this multi-controller architecture is the separation of the security protocol from the network stack. The security protocol used in the project is *Transport Layer Security* (TLS) 1.2[4], implemented by the open source project *mbedTLS*[5], along with the network stack *LightweightIP*[6] licensed under the BSD license. For the MCUs the real-time kernel (version 10.3.1) *FreeRTOS*[7] for resource constrained systems with custom developed low level drivers are used, because they offer a smaller attack surface than e.g. a whole *Linux-distribution*, which has a higher amount of security vulnerabilities.

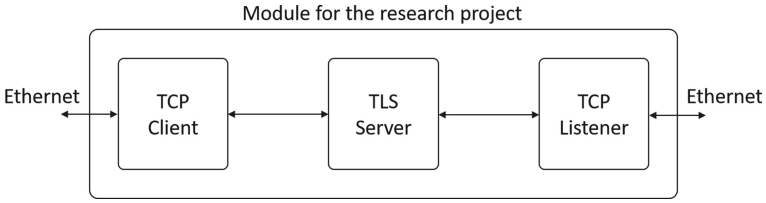

Fig. 3. Task separation of the currently developed research project

4.2 Transmission Speed

The MCU used in our system can run an internal clock speed of up to 480 MHz. This allows us to use the full SPI transmission speed of up to 133 MHz as master [1, p. 193], which results in the fact, that we should be able to realize the aspired 100 MBit/s transmission speed mentioned in Sect. 3. However, it is still necessary to extend the interface, as not all attributes specified are fulfilled by standard SPI.

[3] https://www.st.com/resource/en/datasheet/stm32h743bi.pdf.
[4] https://tools.ietf.org/html/rfc5246.
[5] https://tls.mbed.org/download.
[6] https://savannah.nongnu.org/projects/lwip/.
[7] https://www.freertos.org/index.html.

4.3 Necessary Connections of the Standard SPI

A standard serial peripheral interface has four lines to connect two communication partners. Those lines are the *Clock* (CLK), *Master Out Slave In* (MOSI), *Master In Slave Out* (MISO) and *Chip Select* (CS - sometimes called *Slave Select* (SS)). The last one is necessary when SPI is used as a bus system, where more than two controllers span the communication network. This is not intended for our project as we do only have two controllers. The resulting Point-To-Point (PTP) connection makes this line unnecessary for our concept.

4.4 Equal Transmission Rights

Another special characteristic, which is used within our system, is the so called unconventional *Multimaster SPI*. Using this setting, we do not have a distinct master of the SPI bus, like it is described in Sect. 3. Instead, the master of the communication is exchangeable to be always the transmitter of a message. At the same time, the receiver of data always has to be the SPI slave. This idea is visualized in Fig. 4. The advantage of this principle is an *equal transmission right* for both controllers, in contrast to the one-sided transmission right with normal SPI.

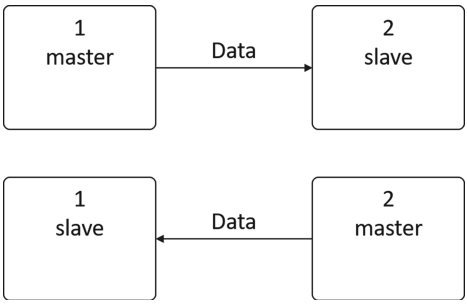

Fig. 4. Distribution of the SPI master attribute for the transmission of data

Simultaneously, when data is always transmitted by the master, there is no need for the Master In Slave Out line, as data will always be put on the *Master Out Slave In* line. This means, the only two connections necessary of the standard SPI communication are the CLK and MOSI lines, which are both controlled by the SPI master and read by the SPI slave. Using this method, the two controllers are always in SPI slave mode, as long as they do not want to transmit data themselves. If they again want to transmit a message, they switch to master mode and write to the lines. The downside of this concept is that there is no longer a Full-Duplex communication possible, as we do only have one line left for data transmission. Furthermore, it needs a kind of flow control to negotiate the transfer right for the prevention of a concurrent writing on the lines.

4.5 Flow Control

While two of the four original SPI lines were removed, additional lines for querying and confirming the transmission right have to be added. These lines get the names *Request To Send* (RTS) and *Clear To Send* (CTS), which are both implemented as *active low*. As the names suggest, the RTS signal of one controller is used to query the right for a transmission to the other controller. The second controller can then allow a transmission via its CTS line. To fully enable this functionality for both controllers, four pins are required, two inputs and two outputs each. The connection principle and line directions (input/output) is shown in Fig. 5. The result of these pins is a *flow control* of the data sent between the two controllers.

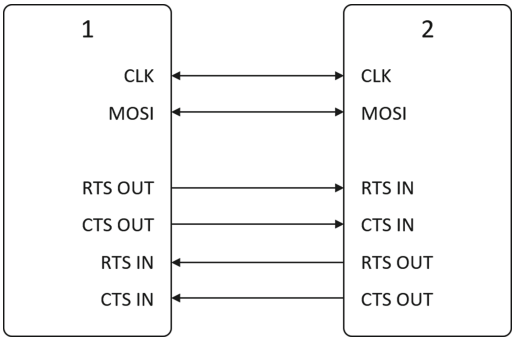

Fig. 5. Connection between the two controllers with additional RTS and CTS input and output lines

Starting a Communication. The procedure depicted in Fig. 6 is required to start a communication. Whenever there is no communication ongoing on the bus, a controller can request a transmission to its communication partner by pulling down its RTS output line. The request can be accepted by the communication partner by simply pulling its own CTS output, which is connected to the CTS input of the requesting controller, to low. This simple principle guarantees that both controllers are set correctly for the next transmission. After this sequence, the SPI master can initiate the transmission by generating a clock signal and putting the corresponding bit sequence onto the MOSI line. At the same time, the SPI slave will read a new bit for every rising edge on the clock pin. Both communication partners do have the information that there is a transmission ongoing due to the flow control mechanism and therefore no data can get lost because of an concurrent write on the lines.

If the requested controller does not acknowledge the transmission with a pull down of its CTS line, an error handling of the message transmission must occur. This can for example be realized by pulling up the RTS line of the requesting

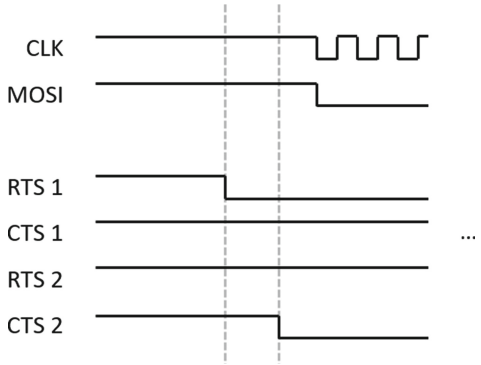

Fig. 6. Procedure for starting a communication

controller, with a following renewed pull down of the RTS line after a defined timeout.

The amount of transmitted bytes in a single transmission has to be set before the transfer itself. There are several ways to solve this task. We have simply added a header with a fixed length, which is sent prior to every payload data. With the constant amount of header bytes, both controllers can set their transmitting/receiving hardware for the right amount of bytes. The header then contains the amount of bytes which will be sent as data after the header block itself. This technique allows both controllers to always have information about the transmission size which is sent between them.

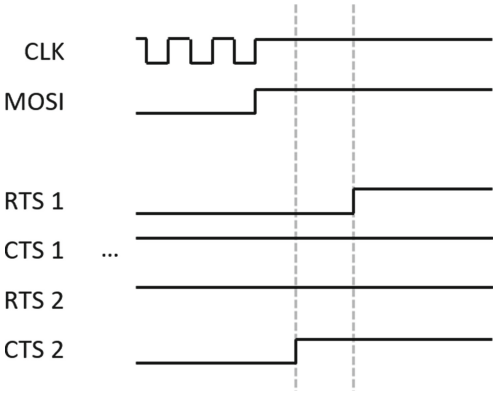

Fig. 7. Procedure for a finished communication

Finished Communication. A data transmission is finished or terminated with the sequence shown in Fig. 7. As both communication partners know the amount of sent bytes, the reading controller confirms the reception of the corresponding

amount of bytes by pulling up its CTS line again. Following that, the sending controller pulls up its RTS line to signal the finished transmission. On the basis of this short sequence, both controllers are informed about the successful transmission.

After this sequence, the transmission of a single data block is finished and both controllers with all lines have reached their initial position. This means, both are ready for a new transmission to be set up.

5 Validation

The first implementation of the introduced protocol of this paper is already finished. We were able to connect two controllers using the multimaster SPI and the additional pins described in Sect. 4. Since we currently do not have a single PCB, to which the three controllers are attached, we are still working with patch wires that connect several NUCLEO boards from STM[8] with each other. This is the reason for using a lower transmission speed of 1 MHz, to avoid damage to the hardware by voltage overshoots on the lines.

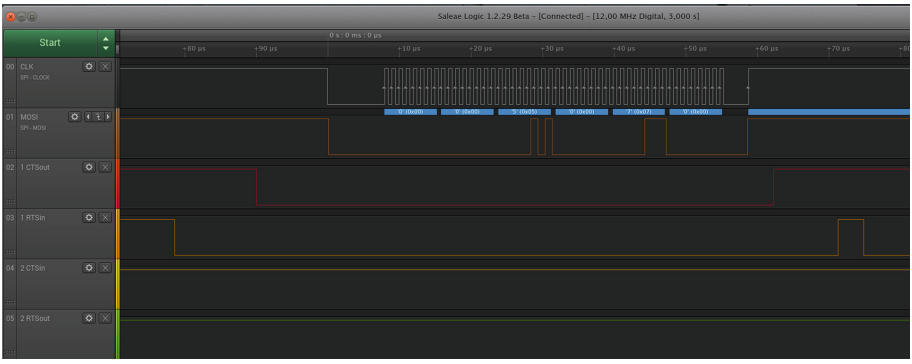

Fig. 8. Extract of an exemplary communication. Communication Partner 1 pulls its RTS line (orange) down to indicate a transmission request. Communication partner 2 replies to the request by pulling down its own CTS line (red) and with that accepting the following transmission. Subsequently the data transmission of the header (6 bytes) with the three 2-byte fields CRC, TYPE and LENGTH can be seen. The termination of the transmission includes the pull up of the CTS line from communication partner 2 with a following pull up of the RTS line of communication partner 1. (Color figure online)

Figure 8 shows the communication between our Point-To-Point connection using a logic analyzer. The described sequence of the corresponding RTS and CTS lines of the two controllers from Sect. 4 is well recognizable. In the figure

[8] https://www.st.com/resource/en/data_brief/nucleo-h743zi.pdf.

we do see a transmission of the mentioned header, which is 6 bytes long in our current implementation. The Header can easily be adapted to any other use case by just changing a few code lines in the software. Our message header is structured as follows. The first two bytes do include a to be implemented Cyclic Redundancy Check (CRC) for our communication. Bytes 3 and 4 include the message type. Those are adapted to the research project and in this case it means that payload data will be sent within the next block. The last two bytes include the data length of the following payload block. The sent data between the two controllers was the simple string *testing*. Every single character is sent as byte (ASCII) which results in the seven payload bytes depicted. The transmission of the data block would include another flow-control scheme described in Sect. 4, but this is not shown in the figure.

As we have not tested the higher transmission speed, this is just a *proof-of-concept*. We plan on making several tests when the PCB with all controllers is finished and ready to use. We will test especially, if we can actually reach the aspired data throughput of 100 MBit/s. However, this will take even more time for investigation and planning on testing and measuring methods and is therefore postponed to a future publication.

6 Conclusion

This paper presents an extended SPI communication interface using multimaster SPI, which was developed during a research project at the Technical University of Applied Sciences in Regensburg. It establishes a Point-To-Point connection of two MCUs, where both communication partners have a fully equal right to write on the bus. To prevent the loss of data because of a concurrent write on the lines, a flow control mechanism was implemented which allows the negotiation of the write permission. A prototype of the concept was successfully implemented, which constitutes the proof of concept. We believe that our prototype can be adopted to many other fields in the embedded world. Detailed load and performance tests are still missing, but will be presented and evaluated in a future publication.

References

1. STM32H742xI/G STM32H743xI/G, 32-bit Arm®Cortex®-M7 480MHz MCUs, up to 2MB Flash, up to 1MB RAM, 46 com. and analog interfaces, Rev. 7, April 2019. https://www.st.com/resource/en/datasheet/stm32h743bi.pdf. Accessed 12 Mar 2020
2. ElPROCUS: Overview on electronic communication protocols (2019). https://www.elprocus.com/communication-protocols/. Accessed 13 Jan 2020
3. Maemunah, M., Riasetiawan, M.: The Architecture of Device Communication in Internet of Things using inter-integrated circuit and serial peripheral interface method. In: 2018 4th International Conference on Science and Technology, ICST, pp. 1–4, August 2018. https://doi.org/10.1109/ICSTC.2018.8528663

4. Niedermaier, M., Merli, D., Sigl, G.: A secure dual-MCU architecture for robust communication of IIoT devices. In: 2019 8th Mediterranean Conference on Embedded Computing, MECO, pp. 1–5, June 2019. https://doi.org/10.1109/MECO.2019.8760188
5. Peng, D., Zhang, H., Li, H., Xia, F.: Development of the communication protocol conversion equipment based on embedded multi-MCU and Mu-C/OS-II. In: 2010 International Conference on Measuring Technology and Mechatronics Automation, vol. 2, pp. 15–18, March 2010. https://doi.org/10.1109/ICMTMA.2010.195
6. Solheim, T., Grannæs, M.: A comparison of serial interfaces on energy critical systems. In: 2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), pp. 1–4, October 2015. https://doi.org/10.1109/NORCHIP.2015.7364373
7. Hill, S.C., Jelemensky, J., Heene, M.R.: US Patent 4816996: Queued serial peripheral interface for use in a data processing system, March 1989. http://www.freepatentsonline.com/4816996.pdf. Accessed 13 Jan 2020
8. Szekacs, A., Szakaill, T., Hegykozi, Z.: Realising the SPI communication in a multiprocessor system. In: 2007 5th International Symposium on Intelligent Systems and Informatics, pp. 213–216, August 2007. https://doi.org/10.1109/SISY.2007.4342659
9. T. Frauenschläger, M. Dentgen, J. Mottok: Systemarchitektur eines Sicherheitsmoduls im Energiesektor, April 2020. https://www.haw-landshut.de/fileadmin/Hochschule_Landshut_NEU/Ungeschuetzt/ITZ_Cluster_Forschung/ClusterMST/Symposium-ESI/2020/Tagungsbandbeitraege/A1-3_OTH-Regensburg_Frauenschlaeger_ESI_2020.pdf. Accessed 12 Apr 2020
10. Tongsan, P., Piromsopa, K.: A software-defined inter-processor communication for embedded system. In: 2016 13th International Joint Conference on Computer Science and Software Engineering, JCSSE, pp. 1–6, July 2016. https://doi.org/10.1109/JCSSE.2016.7748848