Field Bus for Data Exchange and Control of Modular Power Electronic Systems with High Synchronisation Accuracy

Author(s):
Rietmann, Stefan; Fuchs, Simon; Hillers, André; Biela, Jürgen

Publication date:
2018

Permanent link:
https://doi.org/10.3929/ethz-b-000308007

Originally published in:
https://doi.org/10.23919/IPEC.2018.8507631

Field Bus for Data Exchange and Control of Modular Power Electronic Systems with High Synchronisation Accuracy

Stefan Rietmann, Simon Fuchs, André Hillers and Jürgen Biela
Laboratory for High Power Electronic Systems (HPE)
ETH Zürich, Switzerland
Email: rietmann@hpe.ee.ethz.ch

***Abstract*—In this paper, a new field bus protocol based on the IEEE 802.3 Ethernet standard is proposed. It enables fast data exchange between and synchronised control in the modules of modular power electronic systems. With increasing switching frequency, a highly accurate synchronisation of the different modules is a necessary requirement for a control bus. The proposed field bus protocol provides a stable and efficient scheme including a novel data frame structure and allows a synchronisation accuracy of $\pm 4\,\mathrm{ns}$ on the 1 GBit Ethernet standard. For validation of the new protocol a prototype system has been built and measurement results are provided. Eventually, the implementation has been found to meet the specification during testing.**

I. Introduction

In power electronic systems, more and more modular converter structures are applied due to their increased availability (redundancy), simple scaling and improved output quality (e.g. less current ripple by interleaving [1]). The most prominent example for modular converters is the modular multilevel converter (MMC). For the MMC various possibilities for distributed control systems can be found in literature. For example, distributed control methods [2], distributed protection control [3] or modulation methods that allow to distribute the switching signal generation on the modules [4]. As for these distributed control methods, computational power is required on each module/cell (e.g. an FPGA), also more complex communication systems for data exchange and/or communication can be implemented in the modules. Fig. 1 shows an exemplary hardware implementation of such distributed control systems. On each of the modules in the shown MMC stack, there is an FPGA control board connected to an optical communication bus. As can be seen in Fig. 1, the size of the module hardware is increased only very little by the FPGA control board. With a field bus communication system, the number of data connections can be drastically reduced compared to a point-to-point implementation of a master controller to each module/cell. For a daisy chained setup, maximum two lines per slave are required (as e.g. with EtherCAT). The communication system for such modular converter systems typically has the following properties:

1) As all modules are of the same type and require the same communicated data, the individual data frames received and transmitted by the modules are also of the same size.
2) The amount of exchanged data per module is low. It is typically in the range of 5 to 10 Bytes per communication and/or switching cycle.
3) The data exchange on the bus system is bidirectional with respect to all slaves and the master. Every bus member can read from and write to the bus.
4) All modules need to run on the same clock.
5) All modules have to refer to a common point in time, such that they can perform simultaneous or synchronised switching actions (e.g. interleaving). Especially for high switching frequencies, the accuracy of this synchronisation is required to be high (low nanosecond range).
6) Some communication error detection has to be present to protect the system from e.g. implementing wrong reference values.
7) The hardware requirements should be rather low, such that not to much FPGA area is occupied by the communication logic itself rather than the control logic.

In [5], the CAN bus is used to control an MMC. The synchronization accuracy was found to be $\pm 20\,\mu\mathrm{s}$. In [6]–[8] the commercial EtherCAT field bus system has been used

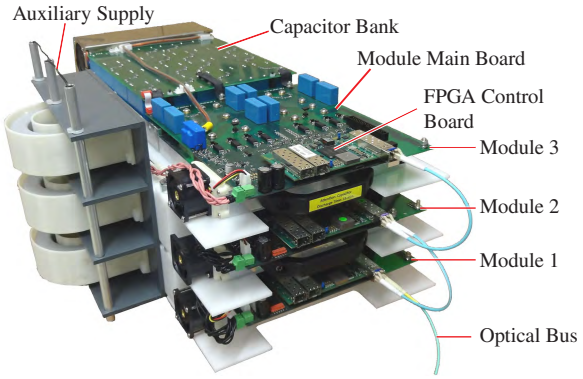

Fig. 1. (a) Photo of a MMC stack (3 modules) using the proposed SyCCo-Bus. It can be seen, that each module is equipped with the same FPGA control hardware. (b) Photo of one MMC modules showing the FPGA control board used for this paper.

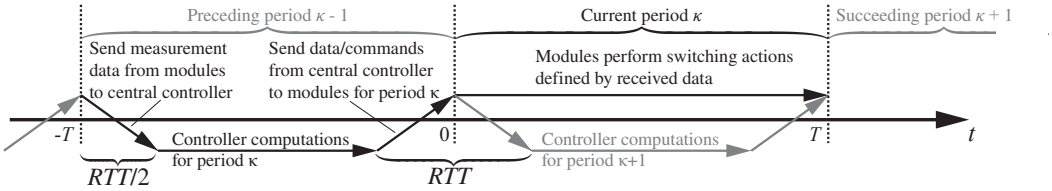

Fig. 2. Typical control timing for modular converters with a central control unit: The central controller receives measurement data from the modules and performs the controller computations based on this data. After that it sends the resulting commands to the modules which perform the switching actions accordingly. Note, that the communication delay, the round-trip time RTT, of the communication system shortens the time available for the control computations to $T - T_\mathrm{d}$, where $T = 1/f_\mathrm{sw}$. The computations and communication actions relevant for the current period κ in which the switching actions are performed are drawn black.

to control an MMC topology. EtherCAT features a synchronization option, that is claimed to result in a jitter of less than $1\,\mu$s [9]. However, in literature the jitter of the synchronisation signals ranges between $\pm 20\,$ns [8] and $15\,\mu$s [6]. For achieving a more precise synchronisation accuracy of $\pm 5\,$ns, a basic concept of a daisy-chained field bus protocol has is proposed in [10]. It is based on $100\,\mathrm{Mbit}$ Ethernet and dedicated physical layer ICs in combination with an FPGA. The master/slaves are interconnected with bidirectional fiber optic lines such that one interconnection per slave results. The protocol tolerates different frame sizes for the individual slaves connected to the bus system and does not require to have knowledge on the number of slaves prior to the operation. This results in a rather complicated start-up procedure and occupies a rather large amount of logic elements/registers in the FPGAs. Therefore, in this paper a simplified frame structure is proposed in order to reduce the implementation effort and significantly reduce the latency of the protocol from [10]. With the new frame structure the amount of received/transmitted data (the frame size) per slave is equal for all slaves and the number of slaves is known a priori. These assumptions lead to a much simpler, more robust and less logic consuming implementation compared to [10]. The delay introduced by the data passing through each slave is also drastically reduced, because all operations on the frame can be performed within one clock cycle and no buffering of multiple frame parts is necessary. The implemented system including the proposed protocol is referenced as **Sy**nchronous-**C**onverter-**Co**ntrol-Bus (SyCCo-Bus). In addition, the hardware setup was changed to Altera Cyclone V GX FPGAs enabling an all integrated transceiver setup using Altera IP cores. This features the Gigabit Ethernet standard increasing the data throughput.

The paper is organized as follows. Section II describes the various requirements necessary for a bus system in modular converter systems. In section III the applied field bus concept, the operating principles and the novel frame structure are introduced. The individual module time synchronization scheme and the expected synchronization accuracy are explained. Section IV refers to the actual hardware implementation including an FPGA based prototype system. The VHDL implementation of the protocol is described in section V. In section VI the concept is tested and particular timing measurement results are shown. The paper concludes in section VII.

II. Data Exchange Requirements in Modular Converter Systems

For explaining the requirements for the data exchange in modular converter systems, a medium voltage MMC setup with $N = 90$ modules and a switching frequency of $f_\mathrm{sw} = 10\,\mathrm{kHz}$ based on the research work of [11] serves as an example in this section. In an MMC, all modules (slaves) have to submit their capacitor voltages ($12\,\mathrm{bit}$) to the central control unit (master) once every switching period (necessary for PWM modulation [4]). Note, that the PWM modulation are done on the modules itself instead of modulations calculated on a central control unit. For protection reasons it can be reasonable to transmit also current measurements ($12\,\mathrm{bit}$) on the modules (cf. [3]). Furthermore, some information on the semiconductor/module-state ($8\,\mathrm{bit}$) as well as temperature information ($8\,\mathrm{bit}$) could be transmitted from the slaves to the master. All this data sums up to a worst case scenario of $5\,\mathrm{Byte}$ per slave. The central control unit has to distribute data to the modules. The duty cycle for the upcoming switching period ($10 - 12\,\mathrm{bit}$) as well as the max. and min. values for current and module voltage ($4 \times 12\,\mathrm{bit}$). This sums up to $8\,\mathrm{Byte}$ of data per slave. For the presented protocol, both directions have the same frame structure, such that one needs to use $8\,\mathrm{Byte}$ data/module for both directions plus a CRC byte for data integrity check.

The frame has to be communicated every switching period, such that in the considered example a minimum data rate of $N \cdot f_\mathrm{sw} \cdot (8+1)\,\mathrm{Byte} = 65.8\,\mathrm{Mbit/s}$ is required for each direction plus the protocol overhead.

This value does, however, not contain information about the delay introduced by the communication system. The larger the delay, the less time there is for computations on the central control unit (cf. Fig. 2). Thus, the communication delay should be lower than half a switching period for a complete communication cycle. This results in a maximum cycle time of $0.5/f_\mathrm{sw} = 50\,\mu$s which is equal to $550\,$ns per slave or a delay of $69.5\,$ns per slave and byte of data. If one calculates the bus data rate necessary for the time the communication is actively transmitting and receiving a frame, a minimum of $90 \cdot (8+1)\,\mathrm{Byte}/50\,\mathrm{us} = 129.6\,\mathrm{Mbit/s}$ results.

The accuracy of the synchronisation should be higher than the possible implementation accuracy of the PWM generated on the modules. For a $10\,\mathrm{bit}$ PWM, this results in a minimum synchronisation accuracy of $1/f_\mathrm{sw}/(2^{10}-1) = 97.8\,\mathrm{ns}$.

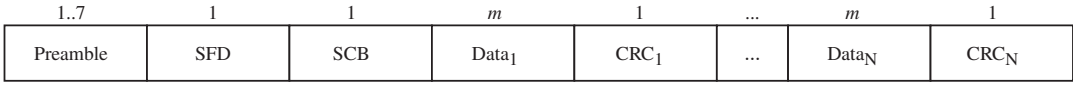

Fig. 3. Frame schematic: The frame is divided into two distinct sections, the header and the data section. The header section mainly contains the preamble and the SFD bytes. Furthermore, the data section consists out of multiple slave specific sections of which each contains a slave data and a CRC section. Note that the number of possible slaves is currently limited to 255 by the current frame structure since the SCB contains $8\,\mathrm{bit}$. This limitation can be solved by increasing the number of SCB bytes.

One can state quite high requirements regarding the data rate for a realistic MMC system. Thanks to the rather low switching frequency in MMC applications, the requirements concerning the synchronisation accuracy and the communication delay are less demanding. Nevertheless, for systems with higher switching frequencies, the synchronisation accuracy must be much higher. In [12], $100\,\mathrm{kHz}$ with the same duty cycle precision ($10\,\mathrm{bit}$) would result in a required accuracy of $9.5\,\mathrm{ns}$.

III. Operating Principles

A. Field Bus Concept

The proposed field bus protocol presumes a daisy-chained network topology. Considering the data flow direction each module is linked to its successor over the forward path and hence the predecessors are attached on the backward path. Compared to a star-like topology this field bus concept does only allow broadcast message transfers, meaning that the complete set of information is always sent to all participants. To initialize a transmission process the master module is issuing the data frame. During the transmission on the forward path each slave module has the chance to read data from the bus and write information back to the bus. A slave being at the end of the forward path chain detects its state as last slave, exchanges information with the bus frame and returns the frame on the backward path to its predecessor.

B. Frame Structure

The frame structure as shown in Fig. 3 and especially the header partition is mainly based on the Gigabit Ethernet frame defined in IEEE 802.3 [13]. In contrast to this standard only the key elements such as the preamble and the start frame delimiter (SFD) octets have been adopted. The destination and address bytes have been omitted since the master is always addressing all slaves. As a replacement for the addressing scheme a frame internal counter byte, the Slave Counter Byte (SCB), is introduced. The complete system is not limited to any number of slaves. Nevertheless, it is worth to note that a slave count exceeding 255 slaves would require a larger SCB section. Due to the delay introduced by each slave, one has to consider the increasing round trip time (RTT) and therefore a decreasing data rate per slave for an increasing number of slaves. The static payload region contains a distinct data region and an additional CRC byte per slave. A complete frame consist of:

- Preamble, 7 octet
- Start-of-Frame Delimiter (SFD), 1 octet
- Slave Counter Byte (SCB), 1 octet
- Data, m byte per slave
- CRC, 1 byte per slave

C. Module Data Exchange

The data exchange between the bus and the slaves is a time critical process since its duration directly contributes to the overall system latency. This latency is denoted in Fig. 4 as $T_{\mathrm{Forward},i}$ and $T_{\mathrm{Backward},i}$, respectively. Achieving a low latency data exchange will result in a high bus throughput. The static structure of the frame allows an efficient and simple data exchange with low latency on the slaves. A finite state machine (FSM) representation of the internal data exchange logic is given in Fig. 5. The complete frame is looped through each slave module. The internal state machine is triggered by the arrival of the preamble and prepares to detect the SFD byte. Note that a slave does not count the number of preamble bytes rather than it checks for the specific SFD pattern. Next, the SCB byte is read and incremented on the fly. From the value previously read SCB value the slaves can estimate their position in the system. Furthermore, the number of bytes per slave are defined in advance and hence the position of the slave specific data in the frame can be calculated after the appearance of the SFD. The module has to remain in the WaitForSlot state until the first byte of its own data arrives. At this point the state changes to Exchange Data. The incoming frame is stored in internal registers and the data which has been prepared for the master module in advance is replacing the read data on the bus. Since all data is looped through the slaves a simple multiplexer logic as illustrated in Fig. 7 is sufficient to change the output between the incoming data frame and the stored, slave specific data. As a final step, the CRC byte which has been calculated during the Exchange Data state is written onto the bus. Eventually, the slave changes back to the Idle state and the remaining incoming data is passed on. The complete frame is received and further transmitted by every module but only the module specific part of the frame is modified.

D. Synchronization

The different modules do not share a common time base nor the same base clock. Furthermore, different boot durations, clock shifts or transmission delays add a non negligible amount of clock jitter to the complete system. As mentioned in the introduction, it is essential for power electronic systems to run synchronous. Therefore, a scheme which provides a common time base and a common base clock is necessary. The proposed protocol aims at a synchronizing accuracy of $\frac{T}{2}$ where T denotes the period.

By synchronizing the modules the internal logic has to rely on the same. Therefore a reference clock has to be shared with every single module. For this purpose Ethernet uses 8b/10b en- and decoding to ensure enough alternating 1 and 0 bits

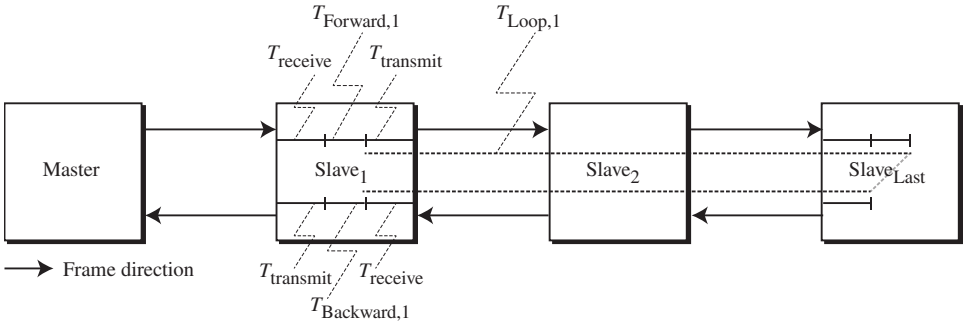

Fig. 4. Synchronisation Schematic: The illustration describes the round trip time measurement of a single slave. Note that the forward path and the backward path are equally long in terms of communication time.

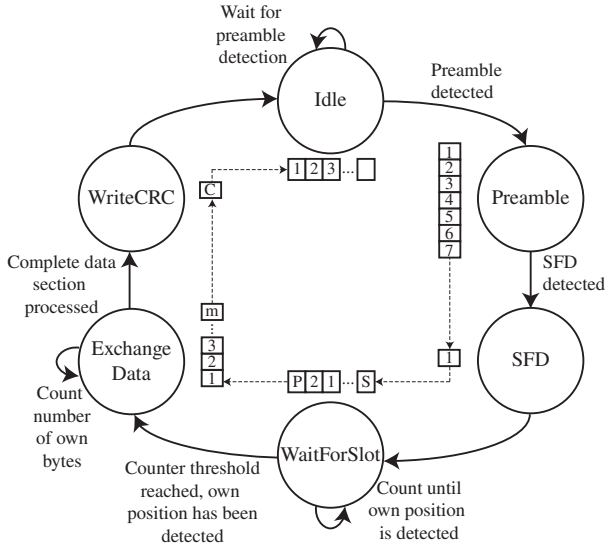

Fig. 5. The Data Exchange Scheme describes the processing of the novel static data frame. The inner illustration shows the number of bytes processed through every step in the complete processing of the data frame.

[13] (cf. Return-to-Zero). This then allows to perform a clock data recovery (CDR) on each module.

With the proposed frame structure, the SFD byte has been introduced as common time token. It is received twice by each slave in each communication round, once on the forward path and once on the backward path. The time measured between the arrival of the SFD on the forward path and on the backward path is referenced as $T_{\text{loop},i}$ as depicted in Fig. 4. The static frame structure as well as the requirement for homogeneous modules allows the conclusion that the transmission time on the forward path is equal to the transmission time on the backward path. Note that the physical wire over which the information is transmitted is the same for the forward and the backward path. In contrast to [10] this allows a simple and accurate determination of a single point in time on which all modules can equally rely.

For this protocol the common time token is used to calculate a waiting time $T_{\text{valid},i}$ for each individual slave. It describes a module specific point in time when every slave has received and processed the slave specific data section of the frame. This means that after detecting the SFD byte, slave i has to wait for $T_{\text{valid},i}$ until the last slave has completely processed the current frame. Hence, it can be used as a starting point for e.g. switching activities in the converter modules. The measured time $T_{\text{loop},i}$ includes the slave internal latency $T_{\text{Forward},i}$ and $T_{\text{Backward},i}$, the transceiver delay T_{receive} and $T_{\text{transmit},i}$ and the physical transmission delay as indicated in Fig. 4.

First of all, each bus module is determining the point in time where the data has been received by each single participant. The modules are then counting the time between the event of sending the SFD byte to the forward path and receiving the SFD on the backward path (cf Fig. 4). Due to the requirement of homogeneous modules and the static frame construction the data processing delay $T_{\text{Forward},i}$ and $T_{\text{Backward},i}$ on each slave can be safely assumed to be equal for coarse synchronization. By introducing the same delay on the backward path as on the forward path the time measured can be divided by two to determine the point in time where the last slave module receives the frame. Again, thanks to the static frame structure it is known by each slave how much time has to be taken into account until the last slave has completely received its data and CRC bytes. This event is denoted by the slave data valid signal (SlaveDValid_S) which is then set high on all modules. Hence, for the T_{valid} we can state:

$$T_{\text{valid}} = \left(\frac{\text{counter value}}{2} + 1 + n \cdot (m+1) \right) \cdot f_{\text{slave}}^{-1} \qquad (1)$$

The measured $T_{\text{valid},i}$ values are used during the next communication cycle. This has two important consequences. First the complete process needs a starting frame which does not transmit vital information rather than a start-up sequence. After the starting frame the synchronisation time is re-evaluated in each frame cycle again. This allows to compensate for possible temperature and environmental effects. Furthermore, the measurements do not depend on each other, therefore no subsequent errors are possible.

IV. Hardware

In order to demonstrate the capabilities of the proposed field bus implementation, a prototype has been developed. The prototype is part of the custom-made high-speed communication and computing platform shown in Fig. 6 (a) which has been designed at HPE, ETH Zurich. The platform is based on an

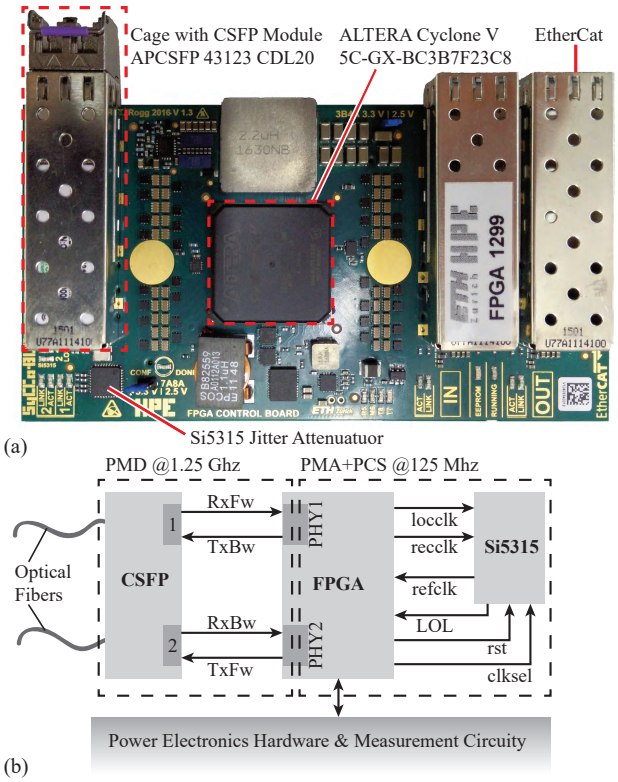

Fig. 6. (a) The developed custom FPGA board including the actual Altera Cyclone V GX FPGA, the transceiver interfaces (CSFP) and a clock jitter attenuator (Si5315). (b) Simplified schematic of the hardware and their interaction signals.

Altera Cyclone V GX FPGA which both runs the firmware of the bus system as well as all specific user-related computation and control tasks involved with the operation of the power electronic systems, for which the platform can be used. In the following, it is briefly explained how the physical layer hardware of the communication system has been designed with an emphasis on a compact realization and how the synchronization of the individual clock domains is achieved.

A. Small Footprint Physical Medium Attachment

In order to keep the board space occupied by the communication hardware to a minimum, the internal multi-purpose high-speed serial transceiver PHY IP cores (cf Fig. 6) of the Cyclone V FPGA are used in conjunction with a common *compact small form factor pluggable* (CSFP) fiber-optic transceivers. The CSFP module presents the physical dependent sublayer (PMD) and attaches directly to the respective inputs / outputs of PHY IP cores of the Cyclone V FPGA using logic-level high-speed differential-mode signaling. The driving of the optics for sending and receiving on the physical medium is handled by the CSFP module.

As opposed to the more widespread *small form factor pluggable* (SFP) transceivers (which work exactly the same way as the CSFP modules), CSFP modules incorporate two independent transceivers in the same form factor. This is achieved by sending and receiving data on different wavelengths of light on a single fiber. The two sockets for fibers of the CSFP module shown in Fig. 6 (a) in fact belong to two independent data channels.

A block-diagram of the physical layer is shown in Fig. 6 (b). The physical medium attachment (PMA) and the physical coding sublayer (PCS) are provided by the PHY IP cores of the Cyclone V FPGA. This way, the space occupied by the physical layer hardware is minimized. Compared to the implementation presented in [10], external physical-layer ICs are thus no longer necessary.

B. Synchronization Hardware

As explained in section III, the presented bus system uses the recovered clock information of the incoming bit-stream as a reference clock for each node, similar to the implementation proposed in [10]. Because the recovered clock is not immediately available at startup, the system boots up with a free running clock and switches hitless over to the recovered clock as soon as the recovered clock is available and stable. The base clock frequency of the transceivers is 125 MHz which is internally multiplied to the 1.25 GHz with which the data is transferred on the medium.

The switchover is handled by the Si5315 jitter-attenuator PLL. In the beginning, the Si5315 starts up with a free-running clock on each board (locclk). This clock is fed back to the FPGA as the reference clock (refclk) for the PMA+PCS as well as the user logic to initiate a link with the previous slave. As soon as the PMA+PCS of the PHY IP core of the Cyclone V FPGA have synchronized to the incoming bitstream of the previous module, the Si5315 is commanded (clksel) to switchover to the recovered receive clock (recclk).

The switchover is performed hitless in incremental steps over consecutive clock periods to prevent a loss-of-lock. The use of an external IC like the Si5315 is neccessary, because the internal general purpose PLLs of the Cyclone V are not specified for transparent clock switchover and do not meet the jitter requirements for clocking the PHY IP cores in this type of application.

V. FPGA Implementation

The proposed SyCCo-Bus protocol has been implemented on the FPGA using the hardware/components described in section IV. The necessary steps to provide a running protocol implementation are described in subsection V-A and V-B.

A. Start-Up

In steady state all slaves run on the master clock that is distributed via the bus. The start-up of the slaves nevertheless has to be executed with a local clock because the clock data recovery (CDR) does not work here yet. To suppress potential oscillations from the CDR during start-up, all slaves do not send any data (and therefore also no clock) to the next slave via their FW path. The master continuously sends a filler frame ('Comma' [13]). At the beginning of the start-up, the first slave synchronizes on the incoming clock recovered from the data received from the master and switches its Si5315 output

to this recovered clock before taking its FW path out of the reset. Now, the second slave is supplied with commas it can synchronize on to recover the first slave's clock which is equal to the master clock. By this mechanism, one slave after the other can synchronize on the master clock before the protocol and therefore the data transmission is started.

B. Protocol Implementation

The complete protocol implementation is differentiated into two distinct modules, the master module and the slave module respectively. As depicted in Fig. 4 the master module only contains a single interface which is connecting it with the subsequent slave module chain.

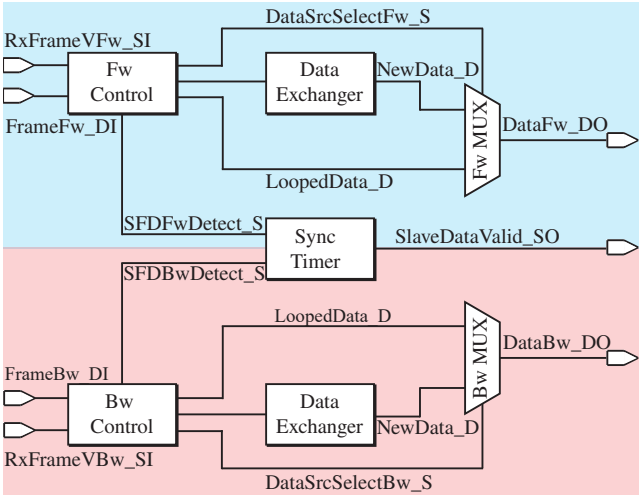

Fig. 7. The slave module is illustrated in a schematic way to show the logical separation between the forward path (blue) and the backward path (red).

In the following the basic structure of the slave modules are explained. Note that the master modules structure is similar to the slave modules structures but includes some more control logic since it initializes the communication rounds.
The modules are differentiated into two separate paths, the forward path and the backward path as illustrated in Fig. 7. Since the transceiver on the FPGA module itself will detect an incoming frame due to the preamble structure the actual slave/master modules can rely on a frame valid signal (RxFrameVFw_SI / RxFrameVBw_SI) issued by the transceiver. Each module then contains a forward and a backward control instance which reacts on this particular valid signal. The control modules are responsible of reading the current byte received from the frame. For the different input bytes such as explained in section 3 the status and control signals of the controller modules output changes. It is especially worth to mention the SFD detect signal which triggers the synchronization process by starting/stopping the counter mechanism. The data exchange sub-module is responsible for the read and write processes on the bus. It is combined with an output multiplexer which is associated with the control module to decide whether the input data frame or the new data gathered from the module has to be sent. The decision is based on state of the current input frame since the a slave module is only allowed to write to its own space in the frame. The data exchange module in the backward is optional as the data does not necessarily have to be exchanged again in the same communication round. Therefore a simplified version of the data exchanger can be used. As a central unit to both paths the synchronization timer triggered by the two distinct SFD detect signals issues a valid signal. The data valid signal is representing a common time token on every module as described in section III-D.

VI. Measurement Results

The measurements have been done using different bus configurations with one master and up to eight slave modules as depicted in Fig. 8. In particular, the synchronisation accuracy has been evaluated to show that the synchronisation method is working as proposed.
The measured round-trip time (RTT) describes the time used by the master to write a complete frame on the bus, send it to all participants and read it back from the backward path. This time leads to the data rate and the data rate per slave which is compared to a theoretical value, the maximum possible data rate.

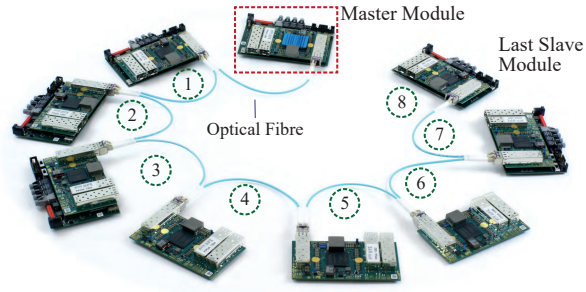

Fig. 8. Exemplary configuration of the SyCCo-Bus with eight slaves and one master module.

A. Synchronisation Accuracy

The synchronisation accuracy has been measured by connecting the internal generated SlaveDValid_S signal to an output pin to observe it with an oscilloscope. Since this signal serves as the common time token for each module over the whole bus, the falling/rising edges observed at the output pins of the different participants are expected to occur at least in the range of one clock cycle ($\pm 4\,\mathrm{ns}$). Fig. 9 shows the synchronisation accuracy measured on a bus configuration with eight slaves and one master. The results show that the current implementation of the bus protocol allows an accuracy of $\pm 4\,\mathrm{ns}$ between the different modules. The measurement has been repeated several times to confirm the shown results, while the configuration has been altered to eliminate potential dependencies between the modules. Also, a long term test with 14 hours run time and more than $4.3 \cdot 10^6$ measurement points has been conducted. The test showed that the signals, and therefore the internal clock signals, are drifting in the range of less than $100\,\mathrm{ps}$. It is important to mention that the synchronisation accuracy inside the stated range is depended on the start up process of the FPGA board and on the

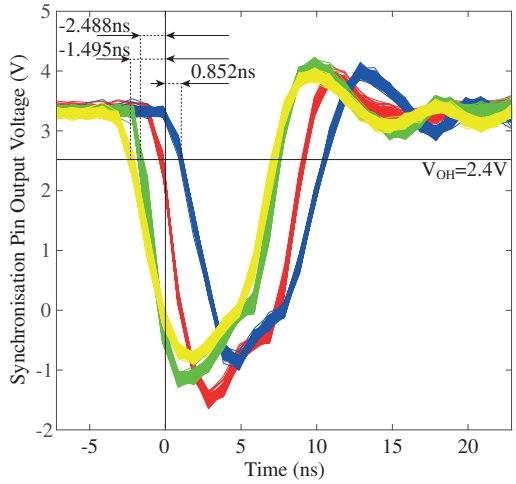

Fig. 9. The measured synchronisation accuracy evaluated over a data set of $2 \cdot 10^3$ trigger points. The signals have been triggered for $V_{OH} = 2.4\,V$. Note that this capture is only exemplary. The distribution of the accuracy in the proposed $\pm 4\,ns$ depends on the power-on process of the FPGA device and the PHY itself.

TABLE I
CYCLE TIME MEASUREMENTS

Bytes/Slave	2 Slaves	4 Slaves	8 Slaves
8	0.984 µs	1.904 µs	3.172 µs
16	1.112 µs	2.152 µs	4.224 µs
32	1.368 µs	2.664 µs	5.248 µs
64	1.880 µs	3.688 µs	7.296 µs
128	2.896 µs	5.728 µs	11.384 µs

individual start up progress of the internal PHY. This leads to a phase difference between the different synchronisation signals which has shown to be non-deterministic. Note that this phase differences denotes the phase differences between the internal 125 MHz Therefore, a simple constant phase offset would not be enough to achieve higher accuracy. Whereas, the long term test showed that almost zero additional phase shift has to be expected and therefore further corrections are only necessary in larger time intervals.

B. Communication Delay

The communication delay has been evaluated in terms of the systems RTT. The RTT is taken as a bare measure to evaluate the delays on the various different participants. The measured RTT for different bus configurations are listed in table I. For a further investigation of the individual delays occurring on the bus, the contributors influencing the RTT are included in (2):

$$\begin{aligned} \text{RTT} = &(2 \cdot T_{\text{Trans}} + T_{\text{Prot}}) \cdot N_{\text{Slave}} \\ &+ T_{\text{Clk}} \cdot (N_{\text{Byte}} \cdot N_{\text{Slave}} + N_{\text{Header}}) \end{aligned} \tag{2}$$

T_{Prot} and T_{Trans} describe the protocol internal and physical delay, respectively. Since the size of the complete frame matters N_{Slave}, the total number of slaves and N_{Byte}, the number of bytes per slave, have to be taken into account. Additionally, N_{Header} is the protocol overhead due to the preamble, the SFD and the SCB octets (cf. section III-B).
All values except for the T_{Trans} are known based on the implementation. In addition those values are adjustable. Whereas, T_{Trans} can only be measured since it includes the physical transmission time via the fibre optical cable and the processing time inside the provided PHY IP and the CSFP modules. The transmission time has been found to be in the range of 390 ns In table I one can see that RTT is almost but not completely doubling, while leaving the number of bytes per slave constant but increasing the number of participants in the system by a factor of two. This is because of the previously mentioned header which acts as a constant offset to the RTT. On the other hand, leaving the number of slaves in the system constant but doubling the number of bytes per slave shows that exactly the added bytes per slave have to be processed.

C. Data Rate

The maximum possible data rate is determined by the underlying link speed but limited by the protocol header. By defining the header and the number of interframe gap bytes (N_{IFG}) one can calculate the data rate achievable on the physical link itself. This can be done by:

$$\text{Data Rate} = \frac{N_{\text{Slave}} \cdot N_{\text{Byte}} \cdot \text{link speed}}{N_{\text{Header}} + N_{\text{Slave}} \cdot N_{\text{Byte}} + N_{\text{IFG}}} \tag{3}$$

To calculate the maximum data rate N_{IFG} has to be equal zero, meaning the the transmission medium is always occupied. Based on the frame structure proposed in Fig. III-B, the data rate has been calculated and is depicted in Fig. 10. In fact, the maximum possible data rate is a theoretical metric since it does not include necessary time gaps between subsequent frames and time delays on the individual modules. Therefore, a comparison with the data rate calculated from the measured RTT has been done as well in Fig. 10. Those values have been evaluated by introducing a minimal interframe gap (IFG) to

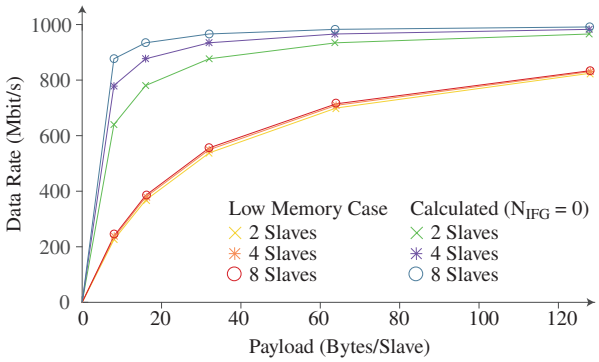

Fig. 10. A Comparison between the theoretical and the measured data rate: The theoretical data rate (blue, green and purple lines) assumes that no gap between two distinct frames is necessary. In fact the protocol will need a certain gap between each frame to reset the every FSM, clean the memory and to make sure that data will not be overwritten. The red, orange and yellow line show the impact of the IFG in the current implementation. Aiming towards a small time gap between two subsequent frames will lead to an substantial increase in necessary memory since every incoming data transmission needs to be stored until its individual SlaveDValid occurs.

satisfy the requirements of the current implementation. Since the data received from the previous frame can not be set valid on any module until the SlaveDValid_S signal occurs, the data needs to be latched. If the IFG should be smaller than the period of the SlaveDValid_S signal, additional memory or registers are necessary to latch more than one single data set.

The comparison between the maximum and the measured values shows, that the influence of the modules individual time delay and the IFG is not negligible. Furthermore, one can see that the data rate increases with increasing frame size and converges to the physical link speed. The more bytes per frame are sent over the channel, the more negligible is the header and a potential IFG. As indicated in section III-B, an

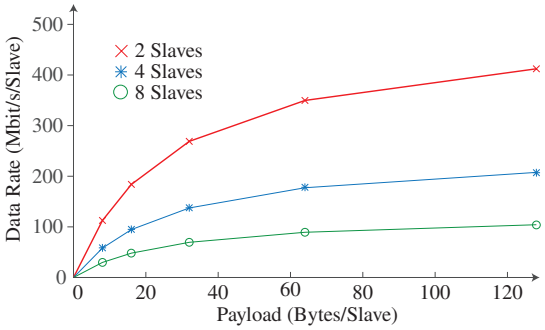

Fig. 11. Data rate per slave: An increasing number of slaves leads to a decreasing data rate per slave, since more participants on the bus will lead to a larger frame. This larger frame eventually results in a longer RTT which means that the individual slaves need to wait longer until they can issue the SlaveDValid_S resulting in a larger IFG.

increasing amount of slaves results in a longer RTT. In Fig. 10 no obvious difference between the different configurations is noticeable since the graph depicts the throughput of the complete bus.

A further interesting and important metric is the data rate per slave, since this is eventually determining how many communication rounds are necessary to complete a message. Fig. 11 shows, that an increasing amount of participants will result in a data rate reduction of each individual slave. With the presented hardware it is possible to run two SyCCo-Bus systems in parallel, as the master still has one free CSFP port. This can significantly reduce the round-trip time (RTT), because the transmission delay introduced per slave ($390\,\mathrm{ns}$) is much larger as the additionally generated overhead ($80\,\mathrm{ns}$). It can thus be assumed that the parallel operation saves approximative half of the RTT.

Of course, one would like to have both bus systems to be synchronous as well. This can easily be achieved by two SyCCo-Bus masters that are implemented on one FPGA on the Master FPGA Board exchanging their measured RTTs (counter value in (1)) such that the difference in their cycle time is known as Δc. The one having the lower counter value has to wait for $\Delta c/2$ cycles before starting to send the next frame, such that the SlaveDataValid signal (cf. Fig. 7) is synchronous on all slaves in both bus systems.

VII. Conclusion

In this paper, the bus protocol for the Synchronous-Converter-Control-Bus-System has been presented. The proposed protocol with the novel static frame structure has been shown to work synchronous in the range of $\pm 4\,\mathrm{ns}$. Furthermore, the implementation of the core of the protocol has been shown and the necessary hardware has been evaluated and built. Eventually, the round-trip time and data rate of different bus configurations have been measured and evaluated. A comparison of the implemented prototype system with a time optimal system has been shown. Furthermore, the prototype system has been found to work for a real world application since the synchronisation accuracy of $\pm 4\,\mathrm{ns}$ and the data rate of more than $700\,\mathrm{MBit/s}$ are both exceeding the necessary requirements of [11]. Since the data rate per slave starts to drop the more slaves are introduced into the system a possible countermeasure has been presented based on the already existing hardware.

References

[1] G. Tsolaridis and J. Biela, "Interleaved Hybrid Control Concept for Multiphase DC-DC Converters," in *IEEE Energy Conversion Congress and Exposition (ECCE)*, Oct. 2017.

[2] M. Hagiwara and H. Akagi, "Control and experiment of pulsewidth-modulated modular multilevel converters," *IEEE Transactions on Power Electronics*, vol. 24, no. 7, pp. 1737–1746, July 2009.

[3] A. Hillers, H. Tu, and J. Biela, "Central control and distributed protection of the DSBC and DSCC modular multilevel converters," in *IEEE Energy Conversion Congress and Exposition (ECCE)*, Sept 2016.

[4] S. Fuchs, S. Beck, and J. Biela, "High output voltage precision PWM for modular multilevel converters," in *19th European Conf. on Power Electronics and Applications (EPE)*, Sept 2017.

[5] M. A. Parker, L. Ran, and S. J. Finney, "Distributed control of a fault-tolerant modular multilevel inverter for direct-drive wind turbine grid interfacing," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 2, pp. 509–522, Feb 2013.

[6] P. D. Burlacu, L. Mathe, and R. Teodorescu, "Synchronization of the distributed pwm carrier waves for modular multilevel converters," in *2014 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, May 2014.

[7] C. L. Toh and L. E. Norum, "Implementation of high speed control network with fail-safe control and communication cable redundancy in modular multilevel converter," in *2013 15th European Conference on Power Electronics and Applications (EPE)*, Sept 2013.

[8] ——, "A high speed control network synchronization jitter evaluation for embedded monitoring and control in modular multilevel converter," in *2013 IEEE Grenoble Conference*, June 2013.

[9] "EtherCAT Technology Group," https://www.ethercat.org, Accessed: 02-26-2018.

[10] C. Carstensen, R. Christen, H. Vollenweider, R. Stark, and J. Biela, "A Converter Control Field Bus Protocol for Power Electronic Systems with a Synchronization Accuracy of ±5ns," in *2015 17th European Conference on Power Electronics and Applications (EPE'15 ECCE-Europe)*, Sept 2015.

[11] A. Hillers and J. Biela, "Increased efficiency and reduced realization effort of DSBC and DSCC modular multilevel converters (MMCs)," in *International Power Electronics Conference (IPEC)*, May 2018.

[12] G. Tsolaridis and J. Biela, "Modular, highly dynamic and ultra-low ripple arbitrary current source for plasma research," in *2017 IEEE 21st International Conference on Pulsed Power (PPC)*, June 2017, pp. 1–4.

[13] IEEE, "IEEE standard for Ethernet," *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–4017, March 2016.