Proceedings of the Federated Conference on Computer Science and Information Systems pp. 665–674
DOI: 10.15439/2018F203
ISSN 2300-5963 ACSIS, Vol. 15

Universal serial bus as a communication medium for prototype networked data acquisition and control systems – performance optimisation and evaluation

Andrzej Tutaj, Jacek Augustyn†
AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics,
Computer Science and Biomedical Engineering
Department of Automatics and Robotics
al. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: tutaj@agh.edu.pl

***Abstract*—Universal serial bus can be considered a cost-effective and high-throughput communication medium for sensor networks and multinode control or data acquisition systems, especially for prototyping purposes. In a prototype system, a PC or Mac computer with a general-purpose operating system is often selected as a host or root node for the USB bus and it acts as a central data collector, supervisory user interface, and network traffic scheduler. However, achieved communication performance is often unsatisfactory since USB stack drivers incorporated in Windows, Linux, or macOS operating systems are not optimised for such specific purposes. The paper shows how an appropriately selected and implemented user application communication schedule, making use of operating system drivers pipelining and multitasking capabilities, can substantially improve USB network throughput and reduce communication latency.**

Keywords*—*universal serial bus; sensor network; distributed and networked control and data acquisition systems; rapid prototyping; communication scheduling; USB stack pipelining and multitasking.

I. Introduction

A COMMUNICATION channel constitutes a crucial part of every sensor network, distributed data acquisition system, or a networked control system. Its performance affects the overall system quality of service. For measurement data acquisition solutions, which often process large streams of data, the most important network characteristic is its throughput. For closed loop control applications the most critical parameter is the round trip time as it directly influences the net loop time delay. There are various networks and protocols available with different properties and characteristics. They differ in popularity, openness, initial costs and implementation efforts.

For small-scale distributed control and data acquisition systems, a full-speed variant of the Universla Serial Bus (USB) 2.0 can be considered an attractive and convenient choice, especially well suited for prototyping purposes. It provides low-cost, high-throughput communication channel with favourable performance-to-price ratio. The network infrastructure can be built using inexpensive and easily available hardware components. Software USB stacks and drivers are readily available for most common general-purpose operating systems (OS), like Windows, Linux, or macOS, which can host popular rapid control prototyping (RCP) software engineering tools like MATLAB/Simulink or LabVIEW. Modern microcontrollers (MCU) and system on chips (SoC), on which network nodes are likely to be built, are routinely equipped with a USB device port peripheral with an integrated PHY module. High-performance USB device stacks are usually available free of charge from MCU manufacturers.

Unfortunately, a USB stack incorporated in a general-purpose OS is usually not well suited for measurement data acquisition or real-time closed-loop control systems, since it has been designed and optimised for different applications. Hence, the performance of such prototype configurations can be poor unless special measures are undertaken. The paper shows how an appropriately selected communication schedule can substantially improve throughput and timing characteristics of a multinode USB 2.0 network comprising PC computer running Windows 7 OS and MATLAB RCP tool as a host node and several MCU-based full-speed device nodes. The schedule is realized by a user application coded as a MATLAB M-file script and does not require any modifications to the standard OS USB stack. Hence, it could be easily implemented under any RCP tool and executed by any unprivileged OS user. The solution takes advantage of a pipelined and multitasked processing implemented by the OS USB driver stack.

The topic of USB bus applications for data acquisition or control purposes is relatively popular in the literature where numerous examples of various such systems can be found. However, less work is reported concerning communication performance optimization and characteristics adaptation. Some researchers restrict their investigations to one-to-one communication systems, comprising a single host and a single device node [1], [2], [3], [4], [5]. Their solutions employ either a USB-to-UART adapter [1], an integrated circuit standalone USB device controller connected to an MCU [2], [3], or

This work was supported by AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland, grant No. 11.11.120.396.
† Deceased.

an MCU or SoC device with an integrated USB peripheral module [4], [5]. Performances of such systems, expressed in terms of data throughput or timing properties, are considered in [6], [7], [8], [9]. Other authors present applications of multinode USB networks for various data collecting or control purposes, including industrial systems, home automation, or virtual instrumentation for power monitoring [10], [11], [12], [13], but do not investigate system performances extensively. Such a study can be found in [14], while the problem of nodes synchronisation is addressed in [15], [16]. Some authors propose hybrid solutions with the USB bus connecting a host computer and a single controller of another multidrop bus, like RS485, CAN, or I2C [17], [18].

The abundance of USB control and data acquisition solutions, on one hand, and rarity of extensive performance analysis and communication optimisation recommendations, on the other hand, encouraged the authors of this article to devote their research to the latter topic.

The paper is organised as follows. An introduction with motivations and a literature review has been given in section I. Section II presents hardware and software architecture of the test bench system that has been used to verify effectiveness of proposed communication schedules for network performance improvement. Four different schedules considered in the paper are elaborated in Section III. Results of experiments are given and discussed in section IV. Section V provides final remarks and further considerations and is followed by acknowledgements and a reference list.

II. Hardware and software architecture of a test system

A. Host and device nodes

Main hardware and software components of a test system, built in order to measure communication performances for various polling schedules, are shown in Fig. 1. Their technical characteristics are given in Tab. I. A portable PC computer running MS Windows OS and hosting MATLAB application is used as a host node of the USB network. A user application responsible for polling all device nodes is coded as an M-file script running in MATLAB environment. The standard USB driver stack of the OS is employed and standard OS Application Programming Interface (API) is used. Device nodes are implemented on an MCU with integrated USB device port using C++ language and bare metal programming approach. A software USB device stack provided by the MCU manufacturer is employed, however some modifications of the stack code for latency reduction are implemented. In a real application, the device node is expected to interface with a physical system being controlled or monitored. However, for communication performance evaluation, this system functionality is irrelevant and has been omitted.

B. USB transfer mode and speed selection

Out of three possible transmission speeds offered by the USB 2.0 specification: low (LS), full (FS), and high (HS), the full speed is a reasonable choice for moderately demanding applications. The data rate is relatively high compared to CAN, RS-485, or similar standards, and the hardware implementation on the USB device side is simplified, as most modern MCU-s and SOC-s incorporate complete full-speed USB peripheral modules. Hence, the FS variant has been selected for USB devices in the study presented in the paper.

TABLE I
Hardware and software components of the test bench

Host node	Hardware	DELL Latitude E6400, Core 2×2.54 GHz, 4 GB RAM notebook PC computer E-Port Plus PRO2X docking station
	Software	MS Windows 7 Professional SP1 operating system CDC USB class driver ver. 6.1.7601.17514 USB host controller driver ver. 6.1.7601.17586 MATLAB ver. 7.9.0.529 R2009b rapid development environment
Device node	Hardware	Olimex SAM7-EX256 Rev. A evaluation board Atmel SAM7X microcontroller based on ARM7TDMI core, 48 MHz
	Software	USB device stack framework streamlined by the authors system-less bare-metal application written by the authors
USB hub	Hardware Software	USB 2.0 high speed hub

There are four transfer modes available with the USB 2.0 protocol: control, interrupt, isochronous, and bulk [19], [20]. Of these, the bulk mode is a natural choice for a distributed system transferring potentially a large amount of data. Unlike the isochronous one, it provides error detection and correction features. Number of transactions allowed in a single USB frame is not limited as with interrupt mode. Although there is no bandwidth reservation for a bulk transfer, it can consume up to 100% of the available bandwidth, provided that there are no other modes transfers scheduled. Large allowable data packet size helps to reduce transmission overhead and thus allows high data throughput.

A standard and popular Communication Device Class (CDC) has been selected for the test application. Software drivers for CDC class are routinely incorporated in most OS-es, making it attractive for rapid prototyping purposes, as no extra programming is required from the user. Virtual Com Port (VCP) driver is used on the host site. It allows standard OS API as well as standard MATLAB functions set for serial port handling to be employed.

C. Data and control flow in the system

Fig. 2 shows relations between MATLAB API function calls, OS API function calls, USB bus transactions, and device node actions. A user M-file script implements polling policy with each individual device node contacted once in a single cycle. The cycle is repeated endlessly. MATLAB *serial* object as well as *fwrite* and *fread* functions are used. Their calls are translated into *write* and *read* OS API calls and interact with the OS USB stack via VCP and CDC drivers. Hardware host and device controllers on the PC and MCU side, respectively, as well as USB 2.0 hubs are engaged in data transfer over

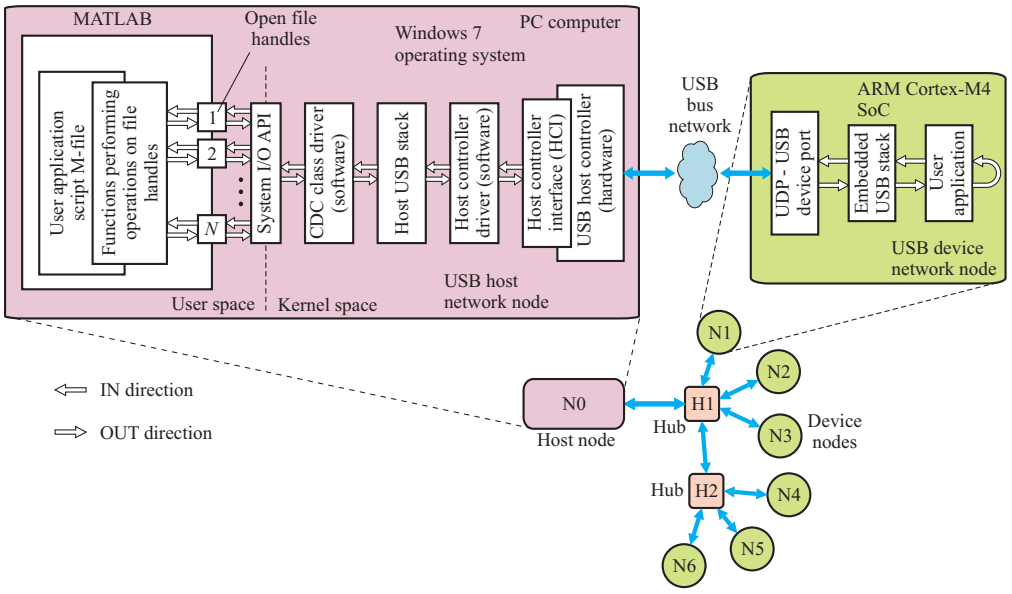

Fig. 1. Hardware and software architecture of the test bench system.

the network. The embedded MCU application on the device side responds to each host query with a predefined amount of data. Greyed and dash-dotted components to the right in the figure, usually present in a real systems, are omitted in the test configuration. The USB device stack provided by the MCU manufacturer has been streamlined by the authors to reduce latency it introduces. It helps to focus the performance study on the network rather than device properties.

In a real control or measurement acquisition system the host is supposed to send to the device control values to be fed to a control plant or parameters controlling the measurement process. The device, on the ohter hand, sends to the host measurement results. One can expect data size asymmetry between *write* and *read* operations with small units of data being sent to the device and large amounts of data being received due to a multichannel or high speed measurements. This expected asymmetry has been taken into account during tests presented further in the paper.

III. User Application Polling Schedules

Four different schedules of device nodes polling by the host side user application are investigated in the article. They are defined, explained and named in the following subsections.

A. Direct interleaved schedule.

Arguably the simplest and the most natural polling scheme is presented in Fig. 3. The user application running on the host node uses *write* call to send a query to a device node and then calls *read* function to wait for a reply. As soon as the data arrives, the host proceeds to the next device. Having finished a full cycle, the host begins a subsequent one. Let us call the duration of a single cycle the *network repetition time* (NRT). It will be used for communication performance evaluation. We will refer to the presented scheme as *direct interleaved schedule*, as *write* and *read* calls alternate and the scheme does not involve any distinct preparatory stage.

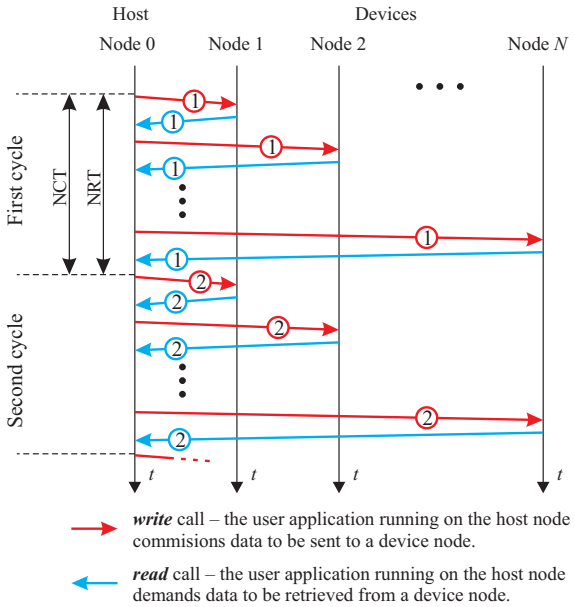

Fig. 3. Direct interleaved schedule – pattern of *write* and *read* I/O functions calls by the user application running on the host node. Encircled numbers on arrows help to match related writing and reading operations (the query and the response corresponding to it).

B. Advanced interleaved schedule

The *advanced interleaved schedule* shown in Fig. 4 differs from the one presented in the previous subsection in having a special initial stage. During this state the host *in advance* writes data to all devices in turn without waiting for any reply. Then it proceeds as with the *direct interleaved schedule*, applying *read* and *write* operations pair to each device in turn

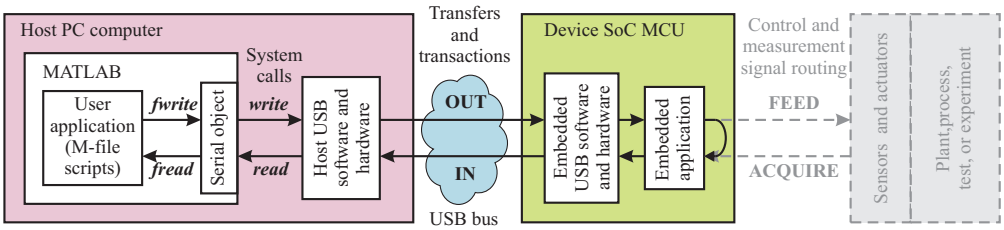

Fig. 2. Relations between user application and OS API function calls and USB bus transaction types.

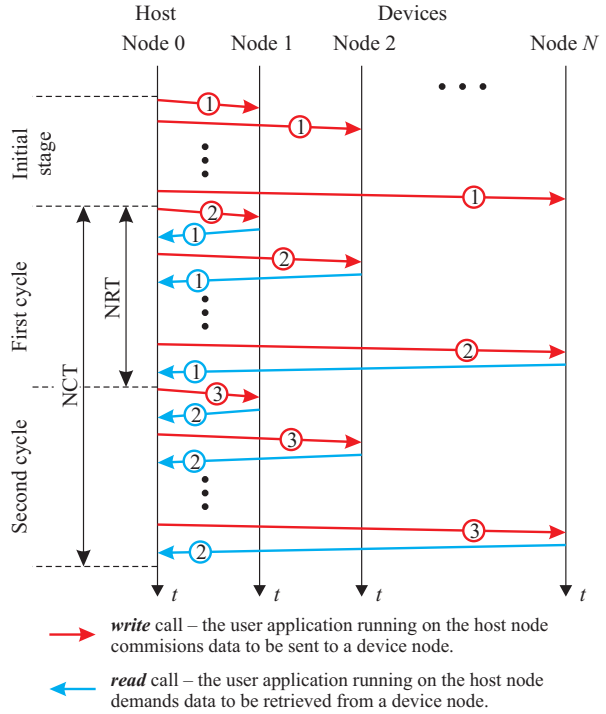

Fig. 4. Advanced interleaved schedule. The *network cycle time* (NCT) equals twice the *network repetition time* (NRT).

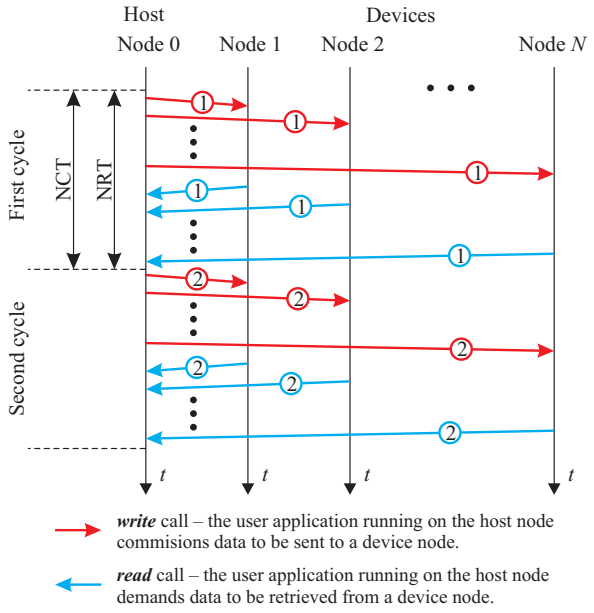

Fig. 5. Direct aggregated schedule. The *network cycle time* (NCT) is equal to the *network repetition time* (NRT).

and repeating the cycle. Thus, there is a sustained excess of writes over reads for each device. It can improve the communication performance by taking advantage of OS USB stack pipelining, buffering, and multithreading capabilities. A new measure called *network control time* (NCT) is introduced in Fig. 4. It is equal to the time elapsing between the beginning of a cycle where *write* operation are effected and the end of a cycle where corresponding *read* calls are completed (note numbers in circles on arrows in the figure). Because of the presence of the initial stage of the schedule, the NCT parameter equals twice the NRT in average. The *control* term in the NCT name alludes to the fact that in a closed-loop control application, NCT rather than NRT parameter influences the quality of control as it contributes to the net time delay in the loop.

C. Direct aggregated schedule

An important drawback of the schedule given in the previous section is that each response received from a device corresponds to last but one query instead of the last one. Hence, there is a one-step query–response shift or delay. Should it be unacceptable for a particular application, one may choose an alternative approach shown in Fig. 5. There is no special initial stage. In every regular cycle the host *aggregates* all write and all read operations in two separate groups, with all writes executed before all reads. That approach provides the stack with an additional time reserve for collection of device replies and does not introduce any shift in messages exchange order.

D. Advanced aggregated schedule.

A combination of *advancing* and *aggregation* techniques is presented in Fig. 6 where the last proposed polling scheme is explained. There is an initial stage comprising *write* calls only while all consecutive cycles start with aggregated writes succeeded by grouped reads. One may expect this schedule to provide further performance improvement by a synergy effect.

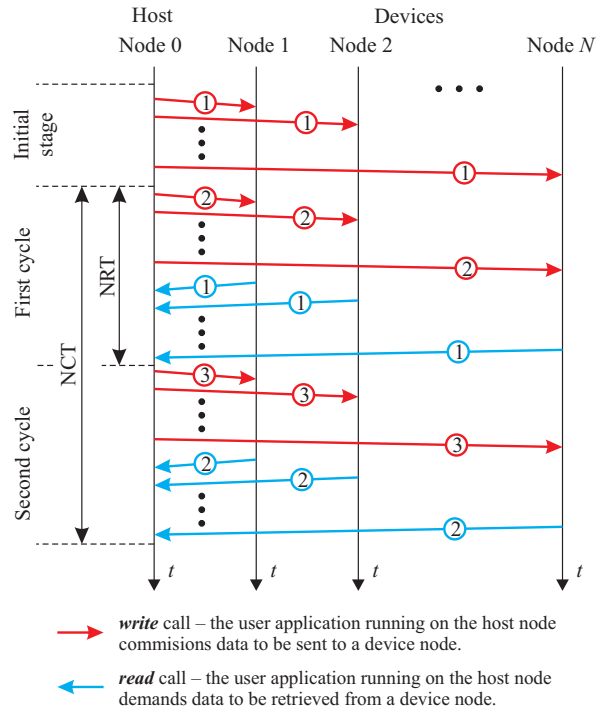

Fig. 6. Advanced aggregated schedule. The *network cycle time* (NCT) equals twice the *network repetition time* (NRT).

IV. Experimental results

A. Test conditions and performance measures

A lot of tests have been conducted for various experimental conditions gathered in Tab. II. Four different polling schedules, proposed in the previous sections, have been investigated in turn. Device nodes number N as well as the data length SIN for a single IN transfer (*read* call) have been varied, while the OUT transaction size SOUT (*write* operation) has been kept constant. A long data series of 10 000 samples have been collected for each experiment in order to compute several statistical performance measures. Four such quantities are used to compare performances of individual scheduling policies, based on network timing or throughput characteristics (see Tab. III). Timing is characterised by NRT and NCT parameters defined in the previous section. They are computed on the host side based on timestamps added to transferred data by device nodes employing hardware peripheral timers. Throughput corresponding to data transferred by IN transactions (*read* operations) is characterised by two related measures: *total network stream* (TNS) and *stream per node* (SPN), satisfying the equation $\mathrm{TNS} = N \times \mathrm{SPN}$ (as long as mean values are considered) where N is the number of active device nodes. For all four quantities their average values (avg) have been computed. For timing related NCT parameter its standard deviation (std) has been also determined. Time series and histograms of timing parameters obtained in selected experiments are presented in the next subsection in Fig. 7–10. Statistics computed from all tests results are gathered in Tab. IV. Discussion of results is also provided. The amount

TABLE II
Experiments conditions and parameters

Parameter or condition	Value or variant
polling schedule	direct interleaved, advanced interleaved, direct aggregated, advanced aggregated
number of active device nodes in the network	1, 2, 3, 4, 5, 6
data length for a single IN transfer (*read* call) SIN, B	48 B, 100 B, 200 B, 500 B, 750 B, 1000 B, 1250 B, 2000 B, 4000 B, 6000 B, 8000 B
data length for a single OUT transaction (*write* call) SOUT, B	16 B

TABLE III
Network performance measures

timing	NRT, ms	network repetition time
	NCT, ms	network control time
throughput	TNS, kB/s	total network stream
	SPN, kB/s	stream per node

of data presented in the table may seem to be intimidating for the reader. However, authors decided to include all results because they share a view expressed in [21]: *It is understood that real time systems are not tested with a single analysis that pronounces them correct. Testing of real time systems is a proof by exhaustion.*

B. Presentation and discussion of experimental results

Fig. 7 presents time series and histograms of the NCT parameter obtained for the *direct interleaved schedule* in an experiment with four active device nodes and IN transfer size of $\mathrm{SIN} = 100\,\mathrm{B}$. For a direct schedule, $\mathrm{NCT} = \mathrm{NRT}$ equality holds. The average (avg) NCT value is equal to $38\,\mathrm{ms}$ (see Tab. IV) and approximately matches performances observed by other authors for USB systems with a single device [7]. Minimum (min) and standard deviation (std) of NCT equal $11\,\mathrm{ms}$ and $21\,\mathrm{ms}$, respectively. Large discrepancy between avg and min value as well as large std/avg ratio reveals a large room for improvement, since the min value estimates the best case scenario. From Tab. IV one can deduce that NCT is approximately proportional to the device nodes number N. That seems to be a natural behaviour for the schedule that executes two-way data exchange witch every devices in turn and does not employ any kind of parallelism. A relation between SIN and NCT is approximately affine with a considerable y-intercept and relatively small slope (NCT increases merely by 60% for SIN increasing over 160 times). It suggests that software components rather than a physical data exchange channel form a communication bottleneck and again suggests a potential for performance improvement by an appropriate polling schedule selection. The TNS is roughly proportional to SIN and almost independent on N. It shows the advantage of using large size transfers and reveals that the available throughput is equally shared by all device nodes.

Time series of NCT and NRT as well as NCT histograms for the *advanced interleaved schedule*, $N = 4$, and $\mathrm{SIN} = 100\,\mathrm{B}$

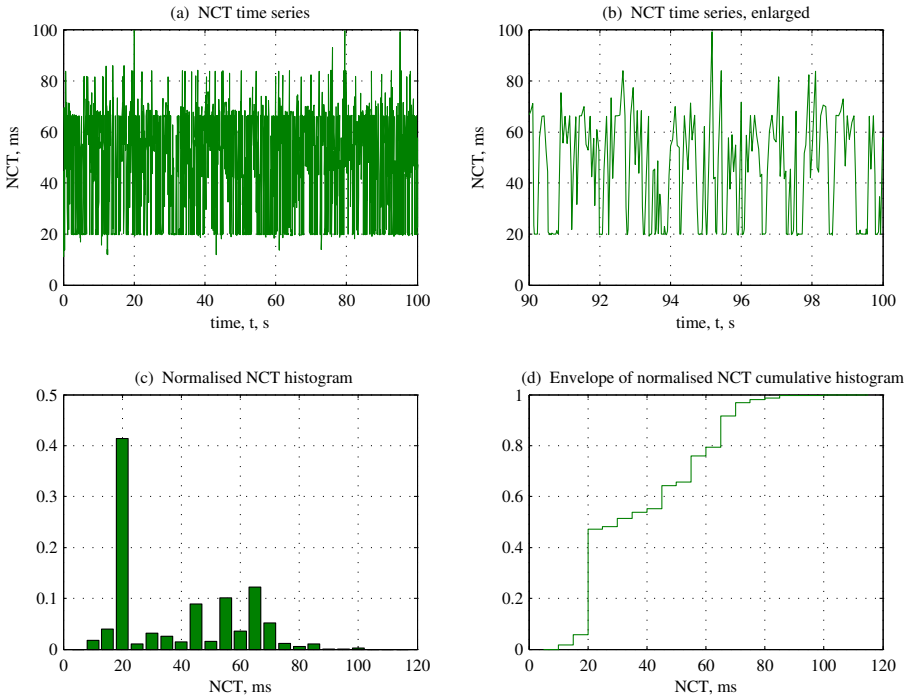

Fig. 7. Results of an experiment for the *direct interleaved schedule* with $N = 4$ active nodes and IN transfer size of 100 B. Time series and histograms of the *network control time* (NCT) parameter.

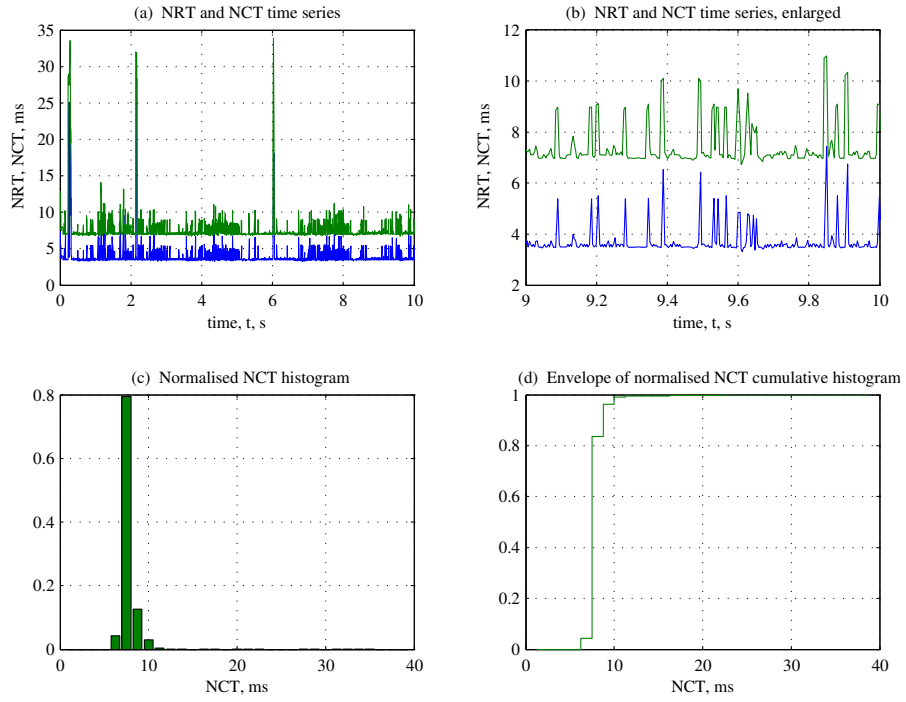

Fig. 8. Results of an experiment for the *advanced interleaved schedule* with $N = 4$ active nodes and IN transfer size of 100 B. Time series and histograms of *network repetition time* (NRT, blue) and *network control time* (NCT, green) parameters.

are presented in Fig. 8. Timing and throughput statistics for various conditions are gathered in Tab. IV. An enormous performance improvement can be observed compared to the *direct interleaved* scheme. The NCT is reduced five times from $38\mathrm{ms}$ to $7.5\mathrm{ms}$. The NRT reduction is even more impressive – about ten times from $38\mathrm{ms}$ to $3.7\mathrm{ms}$. Consequently, both TNS and SPN increase tenfold from $10\,\mathrm{kB/s}$ to $110\,\mathrm{kB/s}$ and from $2.6\,\mathrm{kB/s}$ to $27\,\mathrm{kB/s}$, respectively. The huge improvement is achieved exclusively by introduction of the initial, preparatory stage at the beginning of the cyclic polling schedule (compare Fig. 3 and 4). The one-step shift (and resulting time delay) introduced by the *advancing* technique is by far compensated by the timing performance improvement, revealed by the considerable reduction in both NCT and NRT measures. One can observe that the performance gain is more prominent for small SIN values. Apparently, when the data stream increase, the hardware limitations play more and more important role and diminish benefits brought by the polling scheme modification.

Results for the *direct aggregated schedule* are presented in Fig. 9 and in Tab. IV. For $N = 4$ and $\mathrm{SIN} = 100\,\mathrm{B}$, NCT and NRT become reduced over 6 times (from $38\,\mathrm{ms}$ to $6\,\mathrm{ms}$) compared to the *direct interleaved schedule* while TNS and SPN increase about 6.5 times (from $10\,\mathrm{kB/s}$ to $67\,\mathrm{kB/s}$ and from $2.6\,\mathrm{kB/s}$ to $17\,\mathrm{kB/s}$, respectively). Comparison of two improved polling schemes, the *advanced interleaved schedule* and the *direct aggregated schedule*, reveals that the former performs generally better as long as NRT, TNS, and SPN mean values are considered. However, for the average NCT, the latter scheme shows advantage for most N and SIN combinations. Consequently, for data acquisition systems, the *advanced interleaved schedule* is the preferred one while for the closed-loop control systems the choice should be made based of the number of nodes and IN transfers sizes.

Results of experiments for the *advanced aggregated schedule* are shown in Fig. 10 and in Tab. IV. They reveal a large improvement compared to the *direct interleaved* scheme. On the other hand, the table shows that the performance of this combined schedule is comparable to that obtained for the *advanced interleaved* one. Apparently, the *advancing* approach takes advantage of USB stack pipelinign, multithreading, and parallel computing capabilities to an extent that cannot be further intensified by incorporation of the aggregating technique.

C. Sporadic timing spikes

One can observe sporadic spikes on NCT and NRT time series presented in Fig. 7–10. They are several times higher than the average value of the considered timing parameter. They may result from an occasional lengthy or prolonged preemption of the USB stack or the user application by an unrelated time-consuming task, like hard disk servicing routine. One can expect such behavious since the Windows 7 is not a real-time OS. For a production system such a lack of determinism would be probably a prohibiting factor. For rapid prototyping purposes, however, it may be acceptable, since is far outweighed by development and testing benefits brought by RCP engineering tools like MATLAB or LabVIEW hosted by general-purpose OS-es.

V. Conclusions

The solution presented in the paper is intended for prototype rather than production systems and mainly for rapid prototyping approach. It allows to obtain high performance of a USB-based network despite the application of standard USB stack available in a general-purpose operating system. The dramatic communication improvement is achieved by employment of an appropriately modified read and write function call schedule on the user application side. It takes advantage of a multithreading, parallel computing, buffering and pipelining in the USB stack drivers to streamline the data exchange processes and improve data rate as well as timing characteristics. In a production real-time system designed for data acquisition or distributed control, one may expect protocol stacks to be adapted to the intended applications. That leaves less space for improvement with methods like those proposed in the paper.

All results included in the article have been obtained for a multinode system with several USB devices. However, some proposed methods and schedules can be used as well for a system comprising a single device communicating with a single host. Application of the advancing technique for such a system have been presented in authors' previous work [9].

The paper proves effectiveness of advancing and aggregating techniques in case of a network based on the USB bus technology. However, the authors expect that similar approaches may succeed also for other communication networks and protocols, provided that they make use of a similar software architecture.

The methods given in the article can be beneficial mainly for data acquisition systems, where data throughput maximization rather than closed loop latency (delay) minimization is the main objective. However, to a limited extent, they can also be employed in closed loop control applications as in some cases they also allow reduction of the round trip delay.

Acknowledgements

This work was funded by the AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland, grant No. 11.11.120.396.

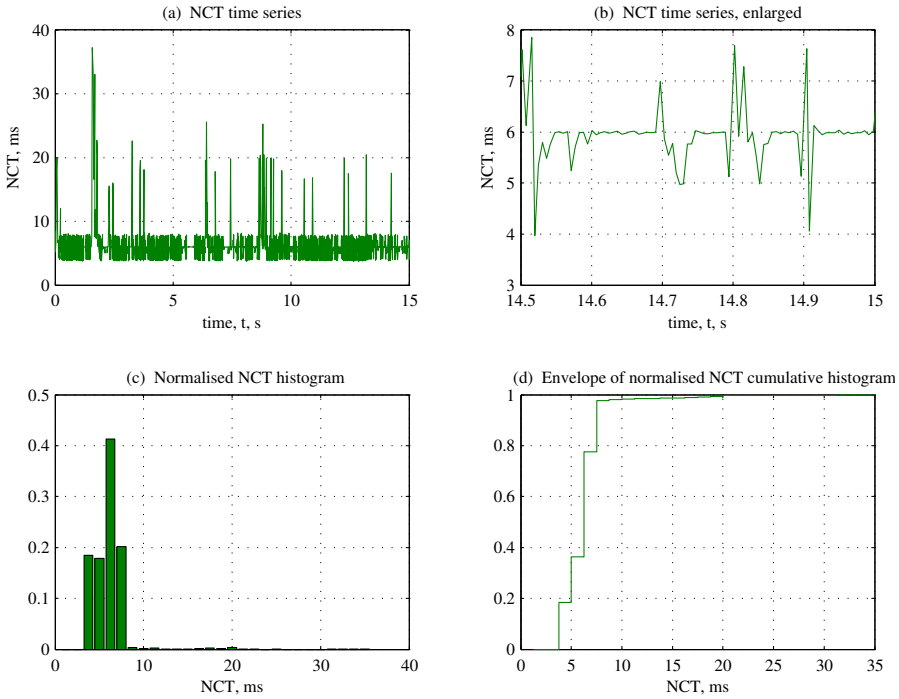

Fig. 9. Results of an experiment for the *direct aggregated schedule* with $N = 4$ active nodes and IN transfer size of 100 B. Time series and histograms of *network control time* (NCT) parameter.

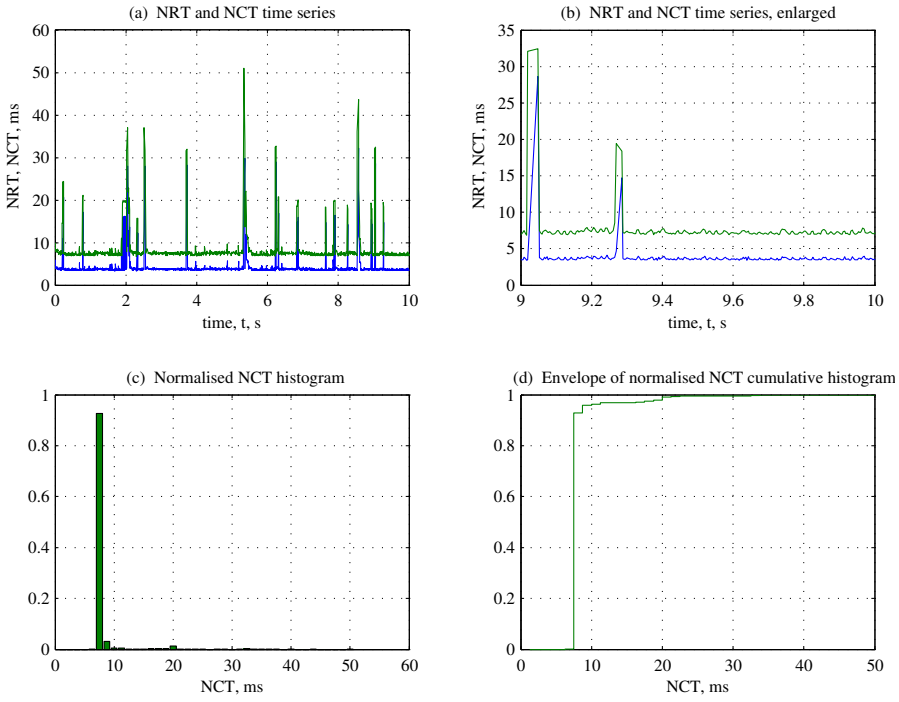

Fig. 10. Results of an experiment for the *advanced aggregated schedule* with $N = 4$ active nodes and IN transfer size of 100 B. Time series and histograms of *network repetition time* (NRT, blue) and *network control time* (NCT, green) parameters.

TABLE IV
RESULTS OF EXPERIMENTS – TIMING AND THROUGHPUT MEASURES STATISTICS FOR DIFFERENT POLLING SCHEDULES AND VARIOUS TEST CONDITIONS. ALL DATA ROUNDED TO TWO SIGNIFICANT DIGITS. RESULTS FOR $N = 4$ AND $\mathrm{SIN} = 100$ B MARKED IN BOLD.

		direct interleaved schedule					advanced interleaved schedule				
	SIN=	48	100	500	2000	8000	48	100	500	2000	8000
NCT avg ms	$N=1$	10	8.3	9.1	13	17	3.6	3.5	4.2	8.8	23
	$N=2$	18	20	25	27	34	4.9	4.9	6	8.8	28
	$N=3$	31	30	38	43	52	6.2	6.3	7.1	11	31
	$N=4$	37	**38**	49	57	69	7.4	**7.5**	7.8	10	28
	$N=5$	54	52	63	70	86	9	9.1	9.4	11	29
	$N=6$	62	57	77	85	100	11	11	11	12	32
NCT std ms	$N=1$	7.6	6.6	7.9	6.9	5.6	1.1	1	1.7	5.2	8.2
	$N=2$	12	12	13	11	8.3	1.1	1.5	0.78	4	9.9
	$N=3$	16	16	17	13	11	1.5	4	1.4	4.4	11
	$N=4$	21	**21**	22	16	13	1.1	**1.5**	5	2.7	9.5
	$N=5$	24	25	26	19	15	1.2	1.9	4.6	2.6	8.7
	$N=6$	29	29	28	22	17	1.2	1.6	1.9	3	6.4
NRT avg ms	$N=1$	10	8.3	9.1	13	17	1.8	1.7	2.1	4.4	12
	$N=2$	18	20	25	27	34	2.5	2.5	3	4.4	14
	$N=3$	31	30	38	43	52	3.1	3.1	3.6	5.3	15
	$N=4$	37	**38**	49	57	69	3.7	**3.7**	3.9	5.1	14
	$N=5$	54	52	63	70	86	4.5	4.6	4.7	5.4	14
	$N=6$	62	57	77	85	100	5.4	5.4	5.5	6.1	16
TNS avg kB/s	$N=1$	4.8	12	55	160	460	26	58	240	450	680
	$N=2$	5.2	10	40	150	460	39	81	330	900	1100
	$N=3$	4.7	10	39	140	460	46	95	420	1100	1500
	$N=4$	5.2	**10**	41	140	460	52	**110**	520	1600	2300
	$N=5$	4.5	9.6	40	140	460	53	110	530	1900	2800
	$N=6$	4.6	11	39	140	460	54	110	550	2000	3000
SPN avg kB/s	$N=1$	4.8	12	55	160	460	26	58	240	450	680
	$N=2$	2.6	5.1	20	74	230	19	41	170	450	560
	$N=3$	1.6	3.4	13	47	150	15	32	140	380	520
	$N=4$	1.3	**2.6**	10	35	120	13	**27**	130	400	570
	$N=5$	0.89	1.9	8	29	93	11	22	110	370	560
	$N=6$	0.77	1.8	6.5	23	77	9	19	91	330	500

		direct aggregated schedule					advanced aggregated schedule				
	SIN=	48	100	500	2000	8000	48	100	500	2000	8000
NCT avg ms	$N=1$	8.8	7.3	8.1	12	18	3.3	3.4	4	8.7	24
	$N=2$	3.3	4.4	9.3	12	20	4.4	4.6	5.1	7	25
	$N=3$	5.2	5.2	6.8	15	23	6.1	6.2	6.2	7.7	24
	$N=4$	6	**6**	6.7	13	25	7.6	**7.9**	8	8.5	20
	$N=5$	7	7.1	7.5	10	26	9.5	9.9	9.6	10	20
	$N=6$	7.1	7.3	8.3	11	30	11	12	12	12	27
NCT std ms	$N=1$	6.6	5.8	7.2	7.5	7.6	1.4	1.6	1.5	5.1	9.5
	$N=2$	2.5	4.2	7.4	7.8	7.3	1.7	2	3.4	2.5	9.8
	$N=3$	3.4	3.2	5	8.9	9.4	2.6	2.5	3.8	3.8	9.4
	$N=4$	2.2	**2.3**	3.3	8.5	8.9	2.2	**2.7**	4.2	2.6	4.6
	$N=5$	3.1	3	3.7	4.8	9.6	2.5	3.1	2.6	2.6	4.9
	$N=6$	3.4	2.9	3.4	3.9	8.3	2.6	3	2.6	3	3.3
NRT avg ms	$N=1$	8.8	7.3	8.1	12	18	1.6	1.7	2	4.3	12
	$N=2$	3.3	4.4	9.3	12	20	2.2	2.3	2.5	3.5	13
	$N=3$	5.2	5.2	6.8	15	23	3	3.1	3.1	3.9	12
	$N=4$	6	**6**	6.7	13	25	3.8	**3.9**	4	4.3	9.8
	$N=5$	7	7.1	7.5	10	26	4.8	4.9	4.8	5.1	9.9
	$N=6$	7.1	7.3	8.3	11	30	5.7	5.8	5.8	6.1	13
TNS avg kB/s	$N=1$	5.5	14	62	170	430	29	59	250	460	660
	$N=2$	29	46	110	340	780	43	87	390	1100	1300
	$N=3$	28	58	220	400	1000	48	97	480	1500	2000
	$N=4$	32	**67**	300	600	1300	51	**100**	500	1900	3300
	$N=5$	34	71	330	1000	1500	50	100	520	2000	4000
	$N=6$	41	83	360	1100	1600	51	100	520	2000	3500
SPN avg kB/s	$N=1$	5.5	14	62	170	430	29	59	250	460	660
	$N=2$	14	23	54	170	390	22	44	200	570	630
	$N=3$	9.2	19	74	130	340	16	32	160	520	660
	$N=4$	8	**17**	75	150	320	13	**25**	120	470	810
	$N=5$	6.8	14	67	200	310	10	20	100	400	800
	$N=6$	6.8	14	60	190	260	8.5	17	87	330	590

REFERENCES

[1] M. A. Ahmad, A. N. K. Nasir, N. S. Pakheri, N. M. Ghani, M. A. Zawawi, and N. H. Noordin, "Microcontroller-based input shaping for vibration control of flexible manipulator system," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 6, pp. 597–610, 2011.

[2] C. Qiong, P. Zhuo, and C. Hui, "The communication design of simulation and measurement for excitation system based on USB2.0," in *2nd International Workshop on Intelligent Systems and Applications (ISA)*, Wuhan, China, 22-23 May 2010. doi: 10.1109/IWISA.2010.5473535 pp. 1–4. [Online]. Available: https://doi.org/10.1109/IWISA.2010.5473535

[3] T. Baohua and Q. Shuhai, "A high speed data acquisition card based on USB bus," in *International Conference on Machine Vision and Human Machine Interface (MVHI)*, Kaifeng, China, 24-25 April 2010. doi: 10.1109/MVHI.2010.179 pp. 357–360. [Online]. Available: https://doi.org/10.1109/MVHI.2010.179

[4] A. Kumar, I. P. Singh, and S. K. Sud, "Energy efficient and low cost indoor environment monitoring system based on the IEEE 1451 standard," *IEEE Sensors Journal*, vol. 11, no. 10, pp. 2598–2610, 2011. doi: 10.1109/JSEN.2011.2148171. [Online]. Available: https://doi.org/10.1109/JSEN.2011.2148171

[5] G. Wang, X. Cheng, and Z. Wang, "Terminal design of the intelligent data acquisition system based on USB interface," *Applied Mechanics and Materials*, vol. 380-384, pp. 3629–3632, 2013. doi: 10.4028/www.scientific.net/AMM.380-384.3629. [Online]. Available: https://doi.org/10.4028/www.scientific.net/AMM.380-384.3629

[6] L. Ramadoss and J. Y. Hung, "A study on universal serial bus latency in a real-time control system," in *34th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1-5, Orlando, Florida, USA, 10-13 November 2008. doi: 10.1109/IECON.2008.4757930 pp. 19–24. [Online]. Available: https://doi.org/10.1109/IECON.2008.4757930

[7] R. P. Gomez, J. J. E. Rodriguez, G. A. Hemandez, and A. M. Sibaja, "USB bulk transfers between a PC and a PIC microcontroller for embedded applications," in *5th Electronics, Robotics and Automotive Mechanics Conference Proceedings (CERMA)*, Cuernavaca, Mexico, 30 September - 3 October 2008. doi: 10.1109/CERMA.2008.21 pp. 559–564. [Online]. Available: https://doi.org/10.1109/CERMA.2008.21

[8] J. Augustyn and A. Bieñ, "Real time performance of USB interface in embedded control and measurement systems," *Przegląd Elektrotechniczny*, vol. 85, no. 7, pp. 1–7, 2009.

[9] J. Augustyn and A. Tutaj, "Evaluation and optimisation of communication performance in a hybrid measurement and control system," *Studies in Informatics and Control*, vol. 23, no. 4, pp. 341–351, 2014. doi: 10.24846/v23i4y201404. [Online]. Available: https://doi.org/10.24846/v23i4y201404

[10] A. Depari, A. Flammini, D. Marioli, and A. Taroni, "USB sensor network for industrial applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 7, pp. 1344–1349, July 2008. doi: 10.1109/TIM.2008.915487. [Online]. Available: https://doi.org/10.1109/TIM.2008.915487

[11] Y. S. Kim, H. S. Kim, , and C. G. Lee, "The development of USB home control network system," in *8th International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, vol. 1-3, Kunming, Peoples Republic of China, 6-9 December 2004. doi: 10.1109/ICARCV.2004.1468839 pp. 289–293. [Online]. Available: https://doi.org/10.1109/ICARCV.2004.1468839

[12] C. P. Young, M. J. Devaney, and S. C. Wang, "Universal serial bus enhances virtual instrument-based distributed power monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 50, no. 6, pp. 1692–1697, December 2001. doi: 10.1109/19.982969. [Online]. Available: https://doi.org/10.1109/19.982969

[13] P. P. Stang, S. M. Conolly, J. M. Santos, J. M. Pauly, and G. C. Scott, "Medusa: A scalable MR console using USB," *IEEE Transactions on Medical Imaging*, vol. 31, no. 2, pp. 370–379, 2012. doi: 10.1109/TMI.2011.2169681. [Online]. Available: http://dx.doi.org/10.1109/TMI.2011.2169681

[14] P. E. Guerrero, I. Gurov, A. Buchmann, and K. V. Laerhoven, "Diagnosing the weakest link in wsn testbeds: A reliability and cost analysis of the USB backchannel," in *Proceedings of the 37th Annual IEEE Conference on Local Computer Networks (LCN 2012)*, Clearwater, Florida, USA, 22-25 October 2012. doi: 10.1109/LCNW.2012.6424085 pp. 934–942. [Online]. Available: https://doi.org/10.1109/LCNW.2012.6424085

[15] J. Dvorak and J. Havlik, "Data synchronization for independent USB devices," in *2011 International Conference on Applied Electronics (AE)*, Pilsen, Czech Republic, 7-8 September 2011, pp. 1–3.

[16] P. Foster, A. Kouznetsov, N. Vlasenko, and C. Walker, "Sub-nanosecond distributed synchronisation via the universal serial bus," in *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS 2007)*, 1-3 October 2007. doi: 10.1109/ISPCS.2007.4383772 pp. 44–49. [Online]. Available: https://doi.org/10.1109/ISPCS.2007.4383772

[17] E. J. Bueno, A. Hernandez, F. J. Rodriguez, C. Girón, R. Mateos, and S. Cobreces, "A dsp- and fpga-based industrial control with high-speed communication interfaces for grid converters applied to distributed power generation systems," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 3, pp. 654–669, 2009. doi: 10.1109/TIE.2008.2007043. [Online]. Available: https://doi.org/10.1109/TIE.2008.2007043

[18] U. Saranli, A. Avci, and M. C. Ozturk, "A modular real-time fieldbus architecture for mobile robotic platforms," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 3, pp. 916–927, 2011. doi: 10.1109/TIM.2010.2078351. [Online]. Available: https://doi.org/10.1109/TIM.2010.2078351

[19] *Universal Serial Bus Specification, Rev. 2.0*, 27 April 2000.

[20] J. Axelson, *USB Complete. Everything you need to develop Custom USB peripherals*, 3rd ed. Lakeview Research LLC, 2005.

[21] M. Hall, "Windows CE 5.0 for real time systems," *Embedded Computing Design*, vol. 3, no. 6, pp. 37–43, 13 November 2005.