Evaluation and Optimisation of Communication Performance in a Hybrid Measurement and Control System

Jacek AUGUSTYN, Andrzej TUTAJ

AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland,
jag@agh.edu.pl; tutaj@agh.edu.pl

Abstract: In a modern hybrid measurement and control system a high-speed data acquisition/execution module communicates with a master management and control unit. A typical solution may involve a System on Chip (SoC) device, like ARM Cortex-M core based microcontroller, linked via USB bus with a Commercial Off-The-Shelf (COTS) appliance, such as a tablet, a smartphone, or a PDA. During prototyping stage the COTS is usually temporarily substituted with a PC computer running a rapid prototyping tool such as MATLAB. The paper presents software design considerations and discusses the results of extensive experiments conducted in order to assess if such a prototyping configuration is able to provide a kind of real-time communication performance comparable to that expected in the target system. An optimisation technique is proposed that considerably improves statistics of throughput, round trip time and packet exchanged frequency and that utilises standard operating system software components.

Keywords: soft real-time, embedded system, microcontroller, USB interface, MATLAB

1. Introduction

The design of modern embedded measurement and control systems is moving towards hybrid solutions [1]. They typically involve two main modules: i) a fast acquisition/execution or measurement and control module, the name depending on the application field, and ii) a master module comprising user interfaces, logging subsystem, and external data exchange links. The former module is connected directly to the controlled or monitored system (plant), measures signals from sensors and feeds signals to actuators. The latter gathers process data, performs its registration, interacts with the user and interfaces with other devices. The control algorithm may be implemented, depending on the adopted methodology, either in the measurement and control module or in the master module. It may be also distributed between them with time-critical direct control realised by measurement and control device and computationally intensive supervisory control delegated to the master module, as it usually runs on a more powerful processor [2].

This study proposes an acquisition/execution module based on a 32-bit microcontroller (e.g. with an ARM/Cortex-M core) to handle analogue to digital converters, binary inputs, outputs, PWM generators, and DAC converters. This module could also perform filtering, parameter estimation, or control tasks. The solution ensures short and stable response times in the order of 1–100 μs [3].

A commercial off-the-shelf (COTS) device, such as a PDA, a tablet or a smartphone is proposed in the literature for the module [1], [4–8]. CTOS devices are typically equipped with data exchange interfaces, such as Wi-Fi, Ethernet, Bluetooth, GSM/EDGE/LTE and interact with a user via a touch-sensitive GUI. A discussion supporting the use of COTS hardware can be found in [5] while a review of designing methodologies is presented in [6]. Software/hardware COTS solutions are even proposed for the demanding military [7] and marine [8] applications.

However, a COTS device, typically a closed hardware solution, imposes a serious difficulty in using RS232 and SPI interfaces often necessary to integrate with sensors or other automation equipment. For this reason a Universal Serial Bus (USB) interface is proposed to link the acquisition/execution and the master modules [1,3].

Rapid prototyping environments, such as MATLAB™ or LabVIEW™, are suggested at subsequent steps of the development of the acquisition/execution module (prototyping, testing and validation of application algorithms). Both environments provide extensive sets of functionalities, including signal processing, identification, and control synthesis packages. They also have versatile capabilities in graphical data presentation, which is an important advantage at the testing stage [9]. Both tools can also be used for measurement results recording.

A question arises whether the Matlab environment is able to ensure the kind of real-time performance and provide data throughput between the acquisition/execution and master modules, similar to that expected in the target solution. Such compatibility between the performance levels at the development and production stages is necessary for the portability reason. Details of the connection performance in target solutions implemented with native language can be found in [10], [1], where a throughput in the range of 600–700 kB/s has been obtained. The use of interpreted scripted languages (e.g. Matlab) at the prototyping stage means that properties thus obtained may differ from these based on a native compiled code.

As the USB communication involves some device drivers, one can expect that the best results can be achieved with dedicated custom ones. However implementation of such drivers is a very difficult and time-consuming task that requires profound knowledge of the operating system. Therefore, the reasonable solution is to use standard components and bundled hardware drivers and to optimise their performance.

The literature presents a wide range of applications of microcontrollers communicating with master computers, used in measurement and control systems. Typically these are 8-bit designs using USB-UART converters [11]. However, these full-speed class converters provide lower throughput than potentially achievable from a USB bus, such as 3 Mbit/s in the cases of [12], which is a limiting factor on the solution. In a detailed study of the properties of a direct USB connection between a PIC 18Fx550 microcontroller and a PC computer various software variants yield throughputs in the order of 32 kB/s [13]. An analysis of USB induced delays of various transaction types shows results in the order of 30–50 ms [14]. Descriptions of solutions based on an ARM core with external USB-UART converters can be found in [15]. A data logging application presented in these articles achieve sampling frequency of 14 kHz for 16-bit data samples, that is not very high result either. Other researchers propose a solution involving implementation of a programme in VB.NET language and subsequent exporting of recorded sensor data into Matlab [16].

The objective of this study is to implement a USB connection using Matlab rapid prototyping tool and standard USB components available in the operating system and to test real-time performance and data throughput properties of such a solution. Another important contribution of the paper is a proposition of an optimisation technique substantially improving the performance of the USB connection. The optimisation aim is to achieve with a rapid prototyping configuration the performance close to this required in a target production system, despite the fact that Matlab uses M-file scripts while a production system is usually coded in C/C++ language.

The paper presents experimental results concerning closed loop time delay and data throughput achievable for communication between the acquisition/execution module and the Matlab environment. Tests have been conducted using hardware based on 32-bit ARM/Cortex-M microcontroller. Efficiency of the proposed optimisation is demonstrated. Results of statistical analyses are presented for various data exchange parameters.

The article is organised as follows. Section 2 presents the proposed structure of the data exchange test system implemented by the authors. Results of extensive experiments conducted for different configurations of the system are gathered, compared and discussed in Section 3. An optimisation of the data exchange scenario, considerably improving the system performance, is proposed in Section 4. Results of experiments are provided and compared to those presented in the previous section. Final remarks and plans for future research are given in section 5.

2. Design and Implementation

The hardware architecture of the considered prototyping solution, schematically depicted in Figure 1, involves an acquisition/execution module and Matlab application. The acquisition/execution module is built on a 32-bit System on Chip (SoC) AT91SAM3U with a Cortex-M3 core [17]. The SoC is equipped with a hardware user device port USB-UDP 2.0 full-speed macrocell, as well as a number of other peripherals, including an ADC converter, an analogue comparator, SPI, I2C, UART ports and a PWM generator. The communication devices can use a multichannel DMA controller to relieve the CPU core from data transmission tasks and free up some

processing power. In the tested solution the DMA handles an ADC converter, an SPI port and UART ports. An alternative could be to use a SoC with ARM7, MIPS, or comparable cores. One can expect that other SoCs provide similar performance since both time delay and throughput are affected mainly by the PC computer operating system (including device drivers) and Matlab application.

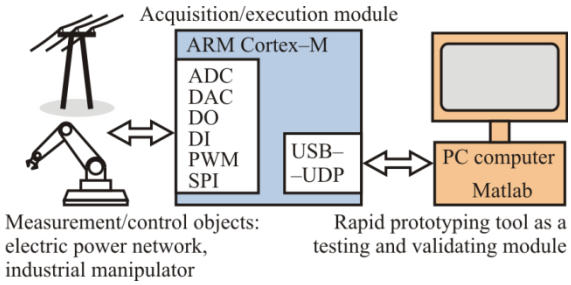

Figure 1. Hardware architecture of proposed system

A Communication Device Class (CDC) protocol has been implemented in the acquisition/execution module for handling USB communications [18]. Two factors determined the choice: bulk transactions type for user data transfer and a wide range of supported operating systems. Indeed, bulk transactions used in this class [19] can be performed several times per millisecond of a USB frame, thus producing faster data streams than control and interrupt transactions, such as in the popular HID class. In case of control and interrupt types the throughput is limited to 64 kB/s because only one transaction up to 64 B is permitted per single 1 ms long USB frame. The choice of available operating systems supporting the CDC class includes Windows XP/7/8, Windows CE/Mobile, Android and Linux, making it useful in target solutions implemented on COTS hardware.

A complete free CDC framework source code is available for the selected processor, but it has been further optimised by the authors to accelerate data exchange handling. An original interrupt service routine (ISR) comprised in the framework has been rewritten to give the precedence to service requests originated by two bulk endpoints (EPs) exchanging user data. Two other EPs (control EP 0 and interrupt IN EP 3) are given a lover priority. Their service requests are handled by the original framework. Bulk transfer EPs requests bypass the stack and are rather handled by a compact code written by the authors. It utilises double hardware FIFO queues attached to the EPs. Right after the reception of an OUT transaction from the host the ISR fills both banks of the FIFO tied to the IN EP. Thus the data is ready when the first IN transaction begins. Then for each consecutive service request the routine refills the emptied bank while the USB peripheral is free to send data from the other one. As the data is always ready, the USB device never answers with NAK to the host. Thus several bulk transactions can be executed within a single 1 ms USB frame, resulting in a large data stream. Otherwise the first NAK sent in an answer to the IN token would cause the host to abandon further attempts till the next frame thus dramatically increasing the transfer time.

A time stamp generated by a hardware timer is added to packages as they are sent. The software has been programmed in C language using a free GNU C compiler. The tests has revealed that the USB interface handling consumes up to 35% of the processor time leaving 40 MIPS (million instructions per second) to execute application signal processing and control tasks.

From the programmer's viewpoint the solution can be modelled with cooperating software layers, as shown in Figure 2.

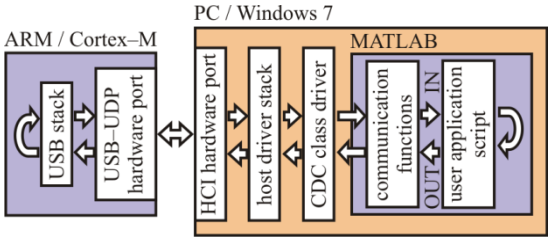

Figure 2. Software architecture of key system components

The user application operating in the Matlab environment works with external layers handling the USB via a number of M-file scripts labelled 'communication functions' in Figure 2. All the scripts are implemented in the Matlab language exclusively without a need for C/C++. They assemble user data packages and issue communication requests to the layer of the CDC driver. A standard driver available in the Windows 7 system for this class is used and it cooperates directly with the USB host driver that sends data to or receives them from the USB bus via a host controller interface (HCI) layer. As response data come into Matlab, the received bytes are converted into floating point values. The user script interprets them and computes the time elapsed between

corresponding sent and received packages (closed loop round trip).

Desktop and laptop PC computers are typically equipped with USB hosts device of the universal host controller interface (UHCI) [20] type as well as an enhanced host controller interface (EHCI) [21] type. Their internal structures differ and hence they are handled by separate device drivers. Both versions has been tested by authors in order to investigate and compare performances provided by these two controller types.

In a PC computer all the connected layers are controlled by an operating system, which can preempt any component for a indeterminate period of time. On the microcontroller side, on the other hand, timing properties can be assessed in a more deterministic manner, provided that no operating system is deployed here and a bare metal programming approach is used instead. Resulting combined performance of the tested system is determined by the interaction between all hardware and software components it comprises.

3. Experimental Results

For the experimental purpose the main Matlab script has been written in such a way that it sends new request package to ARM microcontroller immediately after receiving a response package from it. Thus the package exchange rate is maintained as high as possible. The testing programme monitors the timing of data package exchanges between the user application running in the Matlab environment and the acquisition/execution module.

The total (closed loop) latency is defined as the time delay between the sending of the request and the reception of the corresponding response package. This comprises the assembly of the request package in Matlab, its passage through the USB protocol stack, transfer across the USB link and the reception at the microcontroller plus analogous steps for a response package travelling in the opposite direction. This latency is referred to as RTT (round trip time), while the individual cycle of data package exchange is referred to as a transaction. The presented analysis includes also a statistical parameter referred to as SD and computed using a standard deviation estimation formula that makes it a measure of RTT jitter. The lower the SD value the higher the communication time stability.

Presented test results comprise also the data throughput defined as a mean number of bytes transferred per time unit. The authors decided to consider in throughput measure only the data sent from measurement and control module to Matlab environment. Data travelling the opposite direction are not included as their size in conducted tests is constant and hence their contribution to the total data stream is often negligible.

For typical operating conditions in a target production system packages sent from master module to measurement and control module (OUT) are relatively short at 15–50 B. They contain a coded request for sensor data, but can also include data for executing circuits, such as PWM duty cycles or binary output line settings. The packages received (IN) contain sensor data from the ADC converters and digital input lines. Their length depends on the target application requirements and ranges from tens of bytes to several kilobytes.

Short packages represent typical closed loop control and logging applications where the PC receives sensor data, computes control values, and sends them to the output circuits. Short packages allow shorter RTTs, which is a crucial parameter for the performance of the closed control loop.

In another operational scenario sensor data are captured with a considerably higher frequency, in the order of 20–500 kHz, and then larger packages are assembled and sent from the microcontroller to the PC. This operational condition represents typical measurement and recording applications, which require larger data streams. Also in these applications the master module can estimate object parameters and compute some control commands. Such commands are then send back to the acquisition/execution module within OUT packages with the frequency of transaction repetition, that in this scenario may be several thousand times lover than the frequency of sensor signal sampling.

The experiments for real-time performance and throughput assessment have been conducted for conditions given in Table 1.

During experiments, the operating system runs all of its regular processes and programmes in the background.

Table 1. Experiments conditions

Size of input packages (IN)	48,100,200, *n*·500, 8000 B
Size of output packages (OUT)	16 B
Number of repetitions:	10,000
Computer:	DELL Latitude E6400, 2×2.4 GHz
Docking station:	E-Port Plus PRO2X
Operating system:	Windows 7
Matlab version:	R2009b
CDC driver version:	6.1.7601.17514
UHCI driver version:	6.1.7601.17586
EHCI driver version:	6.1.7601.17586

3.1 Experimental results for UHCI controller

Details of real-time performances for 100 B IN packages and their analysis are illustrated in Figure 3. A UHCI controller achieved an RTT of 9.48 ms. More than 50% of all transaction times fall within 7–9 ms (Figure 3b) and 90.05% of all transaction times are shorter than 10 ms. SD equals 7.29 ms. As much as 90.51% of all transaction are executed in less than (mean+SD).

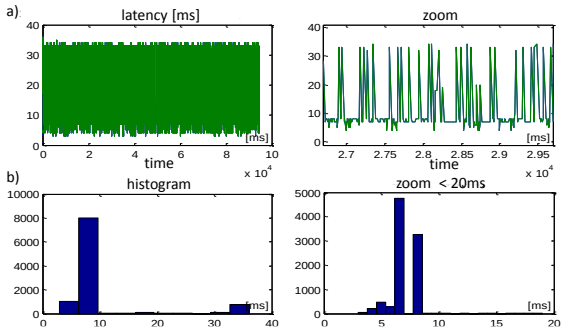

Figure 3. Experimental RTT for UHCI controller. Package size IN=100 B:
a) RTT time series, b) RTT histogram

There are isolated cases of packages transacted at 3 ms, but they constituted a negligible share of the total. About 10% of all packages are transacted at ca. 30–35 ms as a result of unsynchronisation or preemption by the operating system of any of the PC programme components shown in Figure 2. The mean throughput received is 10.55 kB/s, which is a measure of the system capability as a measure-and-record type of application.

Figure 4 illustrates detailed real-time performance with the IN packages size of 3500 B. This increased size extends mean RTT time and SD, but the IN data stream is also considerably larger at 284.3 kB/s. A majority of transactions falls within 9–11 ms and the mean RTT time is 12.31 ms. The remaining details include: 86.3% of the transactions shorter than 13 ms (Figure 4b), 87.99% shorter than 21 ms and about 10% performed within 30–35 ms.

Some isolated cases of times longer than 90 ms are also observed and they result from a preemption of the entire application by the operating system (OS). These occasional cases of very long durations are also observed sporadically in other experiments.

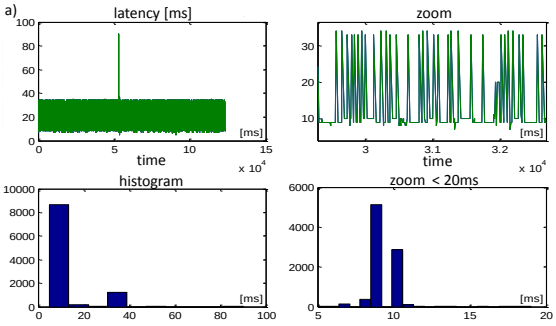

Figure 4. Experimental RTT for UHCI controller. Package size IN=3500 B:
a) RTT time series, b) RTT histogram

A summary of observed RTT statistics depending on package size are illustrated in Figure 5. The graphs show the minimum, mean, mean+SD, max3, and maximum transaction times. The max3 time value is the third longest of all recorded times. It is introduced to exclude some sporadic extreme maximum values thus offering an illustrative view of the occurrence of long delay times.

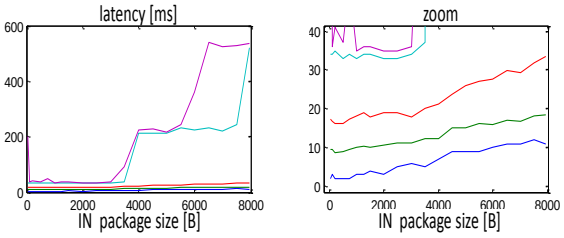

Figure 5. Statistics of RTT experimental results for UHCI controller depending on the package size: minimum (blue), mean (green), mean+SD (red), max3 (cyan) and maximum (magenta)

The minimum can to some extent be thought as a limit imposed by the system hardware/ software configuration. For such interpretation the developer aim can be to make the mean value as close to the minimum as possible. Figure 5 shows abrupt increases of maximum times for packages larger than 4 kB, running up to 200–500 ms, even if this only happens sporadically.

The mean received data stream is illustrated in Figure 6. The stream is heavily dependent on the package size. For example, 1000 B packages are transmitted at 90 kB/s and larger packages are transferred in an even larger streams, up to 425 kB/s (for 8000 B).

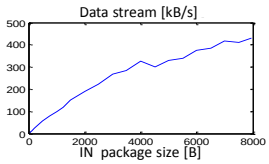

Figure 6. Mean stream throughput in kB/s for UHCI controller depending on the size of incoming package

In these instances all times measured in the experiments increase, e.g. the mean RTT for packages with 8000 B is 18.8 ms (Figure 5).

Experimental results for EHCI controller

EHCI class controllers provide the same bitrate as the UHCI class, but their different hardware design [21] and software drivers allow EHCI to achieve a somewhat higher actual throughput and shorter RTT when compared to UHCI.

Detailed results and their analysis for IN transactions with 100 B packages are shown in Figure 7. The mean RTT is 8.36 ms, i.e. 1.12 ms better than with the UHCI. There is also a significant increase in the proportion of transactions within 4–6 ms (compare with Figure 3b). SD equals 6.67 ms. 82.31% of packages are transacted in less than 9 ms and 86.18% in less than 16 ms.

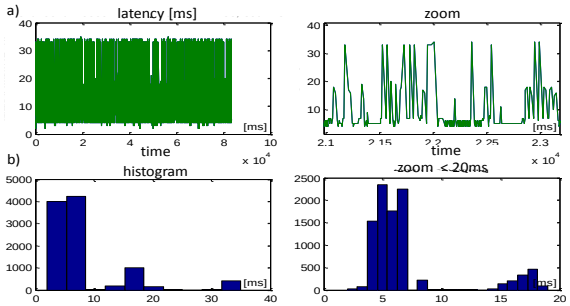

Figure 7. Experimental RTT for EHCI controller. Package size IN=100 B: a) RTT time series, b) RTT histogram

Figure 8 shows a summary of time statistics with an EHCI controller for various package sizes. In comparison to a UHCI the results are better approximately by 1 ms for small packages and by 3 ms for large packages.

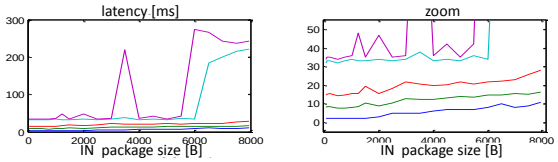

Figure 8. Statistics of RTT experimental results for EHCI controller depending on the package size: minimum (blue), mean (green), mean+SD (red), max3 (cyan) and maximum (magenta)

Mean throughputs obtained with an EHCI (Figure 9) are higher than with a UHCI. For example with an input package of 1000 B the stream throughput is 120 kB/s and for large packages of 8000 B it reaches nearly 500 kB/s. This represents an increase in the order of 20% over a UHCI.

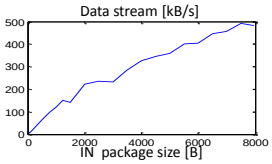

Figure 9. Mean stream throughput in kB/s for EHCI controller depending on the size of incoming package

4. Proposed Optimisation and Its Effect

The performance of the test system given in section 3 is actually poorer than capabilities reported in [1, 10], where all algorithms has been implemented in native (compiled) programming languages. The difference is particularly large in throughput. This means that there exists a potential for optimisation of a connection involving the Matlab environment. Such optimisation is expected to compensate for a performance drop caused by the usage of the interpreted Matlab programming language.

The proposed software optimisation involves concurrent execution of certain part of the package exchange handling code. Since the Matlab environment does not support threads in the script language it is proposed that OUT packages should be queued at a CDC driver level. In this configuration the components: host driver stack – CDC driver – Matlab are capable of exchanging at much shorter waiting time. The proposed optimisation allows parallel execution of certain tasks. Indeed, while one part of a CDC driver is exchanging data with Matlab its other part can be communicating with the host driver. The length of the queue, marked as R, may be equal to one or higher.

The proposed optimisation has also the benefit of requiring no intervention in either the CDC driver code or in its settings.

The optimisation with R=1 increases considerably the mean throughput. The relationship between throughput and package size for the EHCI host are illustrated in Figure 10. The gain in throughput is particularly high with smaller packages, where it increases from 120 kB/s before optimisation to 300 kB/s for IN=1000 B and from 240 kB/s to 566 kB/s for IN=2000 B. The optimum operating condition, suggested by authors, is at around 2 kB, beyond which the gain in throughput is smaller, although packages of IN=8 kB are transacted at 710 kB/s. Packages of the size above 2 kB take significantly longer to transact in real-time.

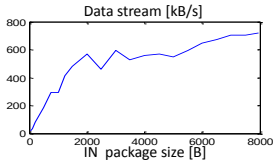

Figure 10. Mean stream throughput in kB/s with optimization (R=1) for EHCI controller depending on the size of incoming package

Let us introduce additional communication quality measure equal to time elapsing between beginnings of two consecutive transactions. Let us call it Transaction Repetition Time (TRT). In Figure 11 an exemplary relationship between RTT and TRT is depicted. Vertical bars located above the time line represents an action of request package commissioning by a Matlab script while bars below the line reflects an event of reply package reception. Curved solid arrows connect corresponding request and response packages. For configuration without package queuing (R=0), RTT and TRT are equal because a new transaction is initiated as soon as the previous one finishes. In case of optimised solution with R≥1, TRT can be considerably shorter than RTT as the next transaction may be initiated before the previous one ends. Thus consecutive transactions can overlap significantly reducing the TRT. For R=1, TRT can be theoretically as low as half the RTT, for R=2 TRT can drop to one third of RTT and so on. One of the main aims of experiments described in this section is to verify if the introduction of queuing can reduce RTT, what conditions are necessary for this to happen, and how large reduction can be achieved.

Detailed real time performance characterisation can be broken down into latency (RTT), repetition period (TRT) and sensor data throughput. Latency is a key parameter for control systems, as it defines the capability of closed control loops to exchange sensor and control data and represents the combined delay of packages (IN OUT) for the same control cycle. Repetition period defines the exchange time of data packages and is important for determination of the speed of subsequent transactions. Sensor data throughput is a key parameter for data acquisition application where large data sets have to be transferred with a high rate.

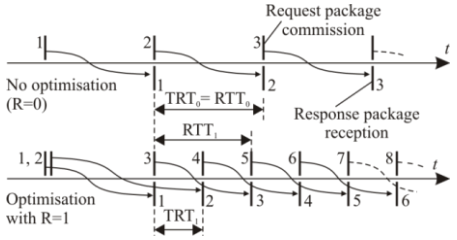

Figure 11. Theoretical relationship between TRT and RTT without and with optimisation (for large IN packages)

In a closed loop control system the RTT contributes to the total control loop time delay while the TRT corresponds to the sampling period. The quality of control usually increases when both time delay and sampling period are reduced. The optimisation proposed by authors can influence both parameters and the change in the quality of control results from a combination of this two factors.

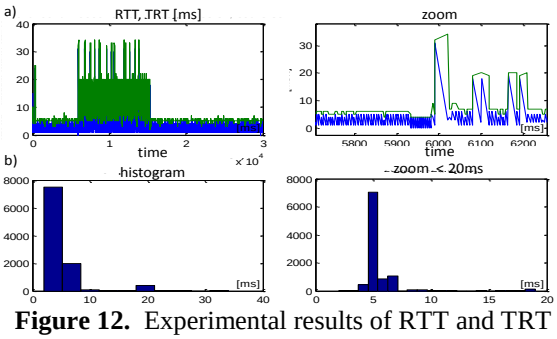

Figure 12. Experimental results of RTT and TRT with optimization (R=1) for EHCI controller. Package size IN=100 B: a) TRT (blue) and RTT (green) recorded, b) RTT histogram

Figure 12 illustrates results obtained for transactions of IN=100 B packages and an EHCI host controller with R=1. The optimisation clearly shortens all data exchange time measures (compare Figures 12a and 7a). The mean latency achieves 5.99 ms, which is significantly lower than that prior to the optimisation (8.36 ms). The SD parameter is also lower at 3.35 ms, which means a better

time stability (lower jitter) than with no optimisation. Latency lower than 6 ms is recorded for 83.5% packages and lower than 10 ms for 95.6% packages. The mean repetition time TRT drops to 2.99 ms and the IN data stream throughput grows to 33.3 kB/s.

For larger packages (IN=2000 B, Figure 13) the mean latency is 7.05 ms and SD equals 2.59 ms. Latency lower than 8 ms is achieved in 88.7% package exchange cycles and lower than 10 ms in 97.22% cycles. The mean data stream throughput is 566 kB/s and the mean repetition time TRT equals 3.53 ms. The ratio of mean values of TRT and RTT is approximately equal to 1:2. This is however not necessarily the case for their instantaneous values as can be clearly seen in Figures 12 or 13. A sporadic increase in the TRT causes the ratio to be temporarily closer to one.

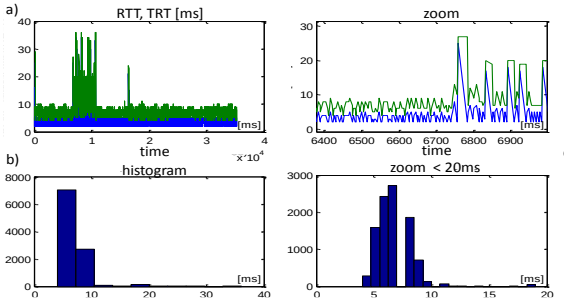

Figure 13. Experimental results of RTT and TRT with optimization (R=1) for EHCI controller. Package size IN=2000 B: a) TRT (blue) and RTT (green) recorded, b) RTT histogram

A summary of real-time properties after optimisation (R=1) is illustrated in Figure 14.

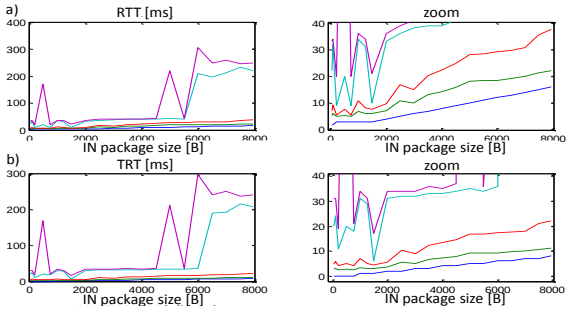

Figure 14. Statistics of RTT and TRT experimental results with optimization (R=1) for EHCI controller depending on the package size: minimum (blue), mean (green), mean+SD (red), max3 (cyan) and maximum (magenta) of: a) RTT, b) TRT

The mean latency for the smallest packages tested, IN=48 B, is 5.37 ms, which is nearly 1.45 times less than without the optimisation. The mean repetition time TRT recorded is 2.69 ms. With packages of IN=4000 B the mean latency equals 14.33 ms and the mean TRT – 7.16 ms.

The data stream performance can be further improved at an expense of latency. Figure 15 presents a summary of throughput results depending on the package size at the optimisation parameter R=3. A data stream throughput of 924.8 kB/s is achieved with the package size as low as IN=3000 B, with the corresponding latency of 12.98 ms and SD=0.66 ms.

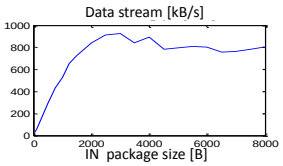

Figure 15. Mean stream throughput in kB/s with optimization (R=3) for EHCI controller depending on the size of incoming package

Details of the real-time performances for packages of IN=1250 B and R=3 are shown in Figure 16.

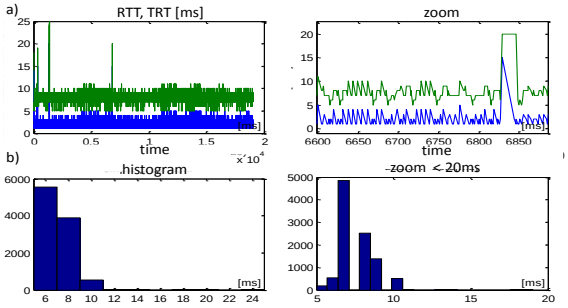

Figure 16. Experimental results of RTT and TRT with optimization (R=3) for EHCI controller. Package size IN=1250 B: a) TRT (blue) and RTT (green) recorded, b) RTT histogram

The data stream is 655.8 kB/s, the mean latency equals 7.63 ms and SD=1.18 ms. This is a far better performance than without the optimisation, both in terms of lower latency variability (lower jitter) and data throughput. The mean repetition time TRT is 1.91 ms.

An exemplary selection of data presented in this and previous sections is given in Tables 2 and 3. They contain only a very limited information compared to several figures included in the discussion, however they can immediately reveal the main tendencies discovered through the research that are important for system designers and developers.

Table 2 shows that an increase in the response package size elongates the RTT but also enlarges the throughput (that is a beneficial result for data acquisition systems). The

difference between system performance with UHCI and EHCI controller are also noticeable in favour of the latter solution.

Table. 2. Selective performance comparison for UHCI versus EHCI controllers without optimisation (TP – IN data mean throughput)

			IN packet size				
			48 B	100 B	1000 B	3500 B	8000 B
Controller	UHCI	RTT, ms	9.47	9.48	11.1	12.3	18.8
		SD, ms	7.48	7.29	8.49	7.91	15.0
		TP, kB/s	5.01	10.6	90.1	284	426
	EHCI	RTT, ms	8.13	8.36	8.33	12.1	16.3
		SD, ms	6.61	6.67	7.25	8.20	12.3
		TP, kB/s	5.90	12.0	120	289	490

Table 3 shows benefits of optimization. Let us compare two rows corresponding to R=0 and R=1 respectively. Optimization technique can increase the throughput over twice (in case of small and medium size packages). For large packages the increase is less pronounced. Counter intuitively, for small and medium size packages, parameter R=1 reduces also noticeably round trip time. It is however not the case for large packages where RTT increases. Optimisation decreases also substantially transaction repetition time. For small packages TRT shortens roughly three times while for large ones the ratio is one and a half.

Table. 3. Selective performance comparison for EHCI controller with and without optimisation (TP – IN data mean throughput)

Optimisation parameter		IN packet size					
		48B	100B	1000B	2000B	4000B	8000B
R=0	RTT, ms	8.13	8.36	8.33	8.33	12.2	16.3
	SD, ms	6.61	6.67	7.25	8.20	8.05	12.3
	TP, kB/s	5.90	12.0	120	240	328	490
	TRT, ms	8.13	8.36	8.33	8.33	12.2	16.3
R=1	RTT, ms	5.37	5.99	6.66	7.05	14.3	22.5
	SD, ms	2.89	3.35	3.71	2.59	9.13	15.4
	TP, kB/s	8.93	33.3	300	566	559	710
	TRT, ms	2.69	2.99	3.33	3.53	7.16	11.3
R=3	RTT, ms	7.50	7.62	7.37	9.50	18.2	39.7
	SD, ms	4.13	1.87	2.43	2.56	4.08	8.53
	TP, kB/s	25.6	52.5	582	842	878	806
	TRT, ms	1.88	1.90	1.84	2.37	4.56	9.92

It is obvious from Table 3 as well as Figure 11 that request package queuing can be beneficial for data acquisition applications as it may increase the throughput twice or even more thus allowing for a higher data collection rate. What is less obvious, the queue may be advantageous also for application with closed control loop. Admittedly, a queue introduces a shift in a train of request packages, but it also causes packages to be sent with a higher frequency. The latter phenomenon can fully compensate for the former or even dominate it (for small and medium size packages). This results in a noticeably shorter RTT (that corresponds to shorter loop time delay) coinciding with considerably shorter TRT (and hence higher sampling period). This is the case for example for EHCI controller with R=1 and IN=100 B. Larger amount of information available to the controller combined with lower time delay in a control loop can result in control quality improvement.

5. Conclusions

The study focuses on an experimental testing of real-time properties of a USB link between a measurement/control system and the Matlab environment. The acquisition/execution module has been implemented with a 32-bit microcontroller based on an ARM Cortex-M3 core and equipped with a hardware USB-UDP port. The study demonstrates that it is possible to achieve a high throughput and low latency with a standard CDC communication class without developing device specific drivers. The proposed optimisation has been proved to deliver the mean sensor data stream in the order of 920 kB/s. This result is comparable to performance achievable in systems where master module is implemented in C/C++ language. Thus, in the proposed solution, Matlab m-file based prototyping environment provides the performance compatible to that expected in a target system.

The entire solution has a universal, low-budget and compact nature thanks to the application of a single SoC microcontroller. Importantly, it allows rapid prototyping, development and validation, as well as laboratory evaluation. The results are applicable for systems implemented on different hardware platforms (with MIPS, AVR32 or FPGA cores).

It is envisaged to extend the research to a USB network comprising several nodes.

Acknowledgements

This work was partially funded by National Centre for Research and Development (NCBiR, Krakow, Poland), grant No. PBS/1/A9/1/2012.

REFERENCES

1. AUGUSTYN, J., **New Methodology of Designing Inexpensive Hybrid Control-acquisition Systems for Mechatronic**

Construction, Sensors, vol. 13(12), 2013, pp. 17222-17240.

2. HAIKAL, A. Y., M. A. ELHOSSEINI, **A Smart Robot Arm Design for Industrial Application**, Studies in Informatics and Control, vol. 23 (1), 2014, pp. 107-116.
3. AUGUSTYN, J., **Design of embedded systems with application to SAM7S family with ARM7TDMI core**, IGSMiE PAN, Kraków, 2007, p. 302.
4. AUGUSTYN, J., A. BIEŃ, **Rapid prototyping methodology of embedded control-acquisition system**, Metrology and Measurement Systems, vol. 19, iss. 4, 2012, pp. 777-786.
5. SALAH, K., M. HAMAWI, **Impact of CPU-bound Processes on IP Forwarding of Linux and Windows XP**, Journal of Universal Computer Science, vol. 16, iss. 21, 2010, pp. 3299-3313.
6. GARCIA, J. Q., **A Study on PDAs for Onboard Applications and Technologies and Methodologies**, Journal Personal and Ubiquitous Computing, vol. 15, iss. 5, 2011, pp. 457-478.
7. WOOLLEY, A., D. DUFF, **Using COTS Technologies for Battlefield Applications**, 2011 Military Communications Confer., Baltimore, 2011, pp. 1506-1510.
8. SHANG, M., X. AIQUIANG, Z. XIULI, Y. CHUNYING, L. TINGJUN, **COST-based Design PMA for Certain Navigation Radar**, 2011 International Conference on Power Electronics and Engineering Application, vol. 23, 2011, pp. 235-240.
9. GUPTA, R., J.N. BERA, M. MITRA, **Development of an Embedded System and MATLAB-based GUI for Online Acquisition and Analysis of ECG Signal**, Measurement, vol. 43, iss. 9, 2010, pp. 1119-1126.
10. AUGUSTYN, J., A. BIEŃ, **Real Time Performance of USB Interface in Embedded Control and Measurement Systems**. Przegląd Elektrotechniczny, vol. 85, iss. 7, 2009, pp. 1-7.
11. AHMAD, M.A., A.N.K. NASIR, N.S. PAKHERI, N.M. GHANI, *at. el.*, **Microcontroller-Based Input Shaping for Vibration Control of Flexible Manipulator System**, Australian Journal of Basic and Applied Sciences, vol. 5, iss. 6, 2011, pp. 597-610.
12. **AN232B-04 Data Throughput, Latency and Handshaking, AN232B-03 Optimizing D2XX Data Throughput**, FTDI Ltd., www.ftdi.com, 2006.
13. POSADA-GOMEZ, R., J. JORGE ENRIQUEZ-RODRIGUEZ, G. ALOR-HERNANDEZ, A. MARTINEZ-SIBAJA, **USB bulk Transfers Between a PC and a PIC Microcontroller for Embedded Applications**, Electronics, Robotics and Automotive Mechanics Conference, Morelos, 2008, pp. 559-564.
14. RAMADOSS, L., J.Y. HUNG, **A Study on Universal Serial Bus Latency in a Real-time Control System**, 34th Annual Conference of IEEE Industrial Electronics, IECON 2008, Orlando, 2008, pp. 67-72.
15. TAN, B., S. QUAN, **A High-speed Data Acquisition Card Based on USB Bus**, International Conference on Machine Vision and Human-machine Interface, Kaifeng, 2010, pp. 357-360.
16. AREFIN, M.M.N., M.N. AMBIA, T. AHAMMAD, A.S.M SHIHAVUDDIN, **Low Cost Design of a PC Based Integrated System for Signal Measurement and Generation using Microcontroller**, 2nd International Conference on Signal Processing Systems (ICSPS), Dalian, 2010, pp. V3-747-V3-751.
17. **AT91SAM ARM-based FLASH MCU SAM3S Series. Doc. 6500C.** Atmel, 2011.
18. **Universal Serial Bus Class Definitions for Communications Devices. Rev. 1.2**, November 16, 2007, www.usb.org.
19. **Universal Serial Bus Specification, Rev. 2.0**, April 27, 2000, www.usb.org.
20. **Universal Host Controller Interface (UHCI) Design Guide. Rev 1.1**, March 1996, Intel, http://download.intel.com/technology/usb/UHCI11D.pdf, 1996.
21. **Enhanced Host Controller Interface Specification for USB**, Rev. 1.0, March 12, 2002, www.intel.com, 2002.