No. 111/19, 7–16
ISSN 2657-6988 (online)
ISSN 2657-5841 (printed)
DOI: 10.26408/111.01

Submitted: 18.06.2018
Accepted: 07.11.2018
Published: 23.09.2019

EMBEDDED WEATHER STATION DATA DISTRIBUTION IN A DATASOCKET CLIENT-SERVER NETWORK ON THE LABVIEW PLATFORM

Dorota Rabczuk

Gdynia Maritime University, Morska 81-87, 81-225 Gdynia, Poland,
Faculty of Electrical Engineering, Department of Marine Telecommunications,
e-mail: d.rabczuk@we.umg.edu.pl, ORCID 0000-0003-0636-0464

Abstract: The article presents a model of a weather receiving station comprising a microcontroller system and LabVIEW application. The microcontroller system is equipped with a receiving radio modem and is intended to decode the frames of data transmitted by a weather station used to monitor weather sensors. The decoded stream of data is sent to a PC computer via a serial interface. The data is distributed over the Ethernet via a LabVIEW DataSocket application, based on the TCP protocol. The data is saved to a text file associated with the local DataSocket server. The DataSocket server allows access to the data by DataSocket clients connecting over the local network. The DataSocket client application enables presentation of current and historical data on a timeline graph.

Keywords: embedded weather station, DataSocket server in LabVIEW, client-server network in LabVIEW.

1. INTRODUCTION

Modern weather stations can be considered implementations of embedded systems. They function on the basis of microcontrollers with sensors on the GPIO (General Purpose Input Output) lines, visualising weather data locally on LCDs or distributing them over wired (e.g. serial) and wireless networks [Popa and Iapa 2011; Shaout et al. 2014]. Communication between the microcontroller and the external user is achieved by TCP data transmission through an Ethernet module connected on the SPI (Serial Peripheral Interface) bus lines and by a radio channel using a radio module in the non-licensed ISM band.

The goal was to combine within a single design an embedded system based on two microcontrollers transmitting weather data over a wireless connection, based on an application developed in LabVIEW, which uses DataSocket components to distribute and visualise the data over an Ethernet network.

The TCP/IP-based DataSocket technology facilitates and automates the establishment of client-server connections. Data are collected on a DataSocket server, accessible by network clients by providing the host URL where the server is running. DataSocket clients that download the data collected on the server are installed on other PCs, forming a distributed data transmission system [Khera and Balguvhar 2013; Yiwei et al. 2017; Pałczyńska and Rabczuk 2018;].

In this design, the data source for the DataSocket server resources is an embedded system that monitors weather measurement sensors on GPIO lines.

2. MICROPROCESSOR-BASED WEATHER DATA RECEIVER

For the purpose of the project, a microprocessor-based weather data receiver was designed and built, equipped with a wireless modem compatible with the purchased radio transmitter weather station and a weather sensor suite installed on a mast. The sensor suite comprised: wind speed and direction (anemometer), temperature, humidity and precipitation sensors.

The weather station wireless modem operated in the 868 MHz band with OOK (On-Off Keying) modulation. Bits were pulse duration modulated (PDM). A logical one was represented by a 0.5 ms positive pulse, while a logical zero was represented by a 1.5 ms positive pulse. There was a 1 ms gap (0V) between the pulses. Pulse duration modulation ensured high pulse width tolerance during readouts, important in high-noise radio channels, while the bi-phased logical states enabled frame auto-synchronisation. Due to the method of bit modulation, frame width expressed in time units was not constant and depended on the percentage share of logical zeroes and ones.

The weather station transmission protocol defined the frame format, which comprised 9 bytes: preamble with device ID, 8 bytes with sensor data, and a single-byte error detection CRC (Cyclic Redundancy Code) [Szlas 2018]. For the purpose of efficient data packaging, the bit sequence in a frame was divided into nibbles (4 bits each). The preamble contained 3 nibbles, temperature with a sign bit was recorded on 3 nibbles, humidity, wind speed and amount of precipitation were recorded on two nibbles, wind direction and low battery signal were one nibble each (Tab. 1).

Table 1. Structure of the weather data frame; abc – device identifier, def – temperature, gh – humidity, ij – average wind speed, kl – wind speed in gusts, op – the amount of rain, q – low battery indication, r – wind direction, st – checksum

Byte (8 bits)	0		1		2		3		4		5		6		7		8		9	
Nibble (4 bits)	a	b	c	d	e	f	g	h	i	j	k	l	m	n	and	p	q	r	s	t

Frames were transmitted cyclically: every 50 s a frame is sent twice with a pause of approx. 30 ms between the frame and its repetition.

Familiarity with the byte modulation method and weather station transmission protocol enabled the design and build of a microprocessor-based receiver station (Fig. 1) with an 868 MHz wireless modem. A PD modulated pulse sequence was downloaded from the modem data line and software decoded by the microcontroller.

The pulse decoding algorithm used two microcontroller interrupts: an external interrupt activated by each pulse slope (descending and ascending) and a timer interrupt that counted equal time intervals of 100 us each in COMPARE CTC (Clear on Compare Match) mode. The 100 us interval was determined by entering 1600 in the timer's OCRxx comparative register, assuming that the microcontroller had a quartz-regulated frequency of 16 MHz, and the timer prescaler value was 1. During the timer interrupt procedure, counter L was incremented, which stored the number of timer interrupts, which in turn was a multiple of the 100 us interval (Fig. 2).

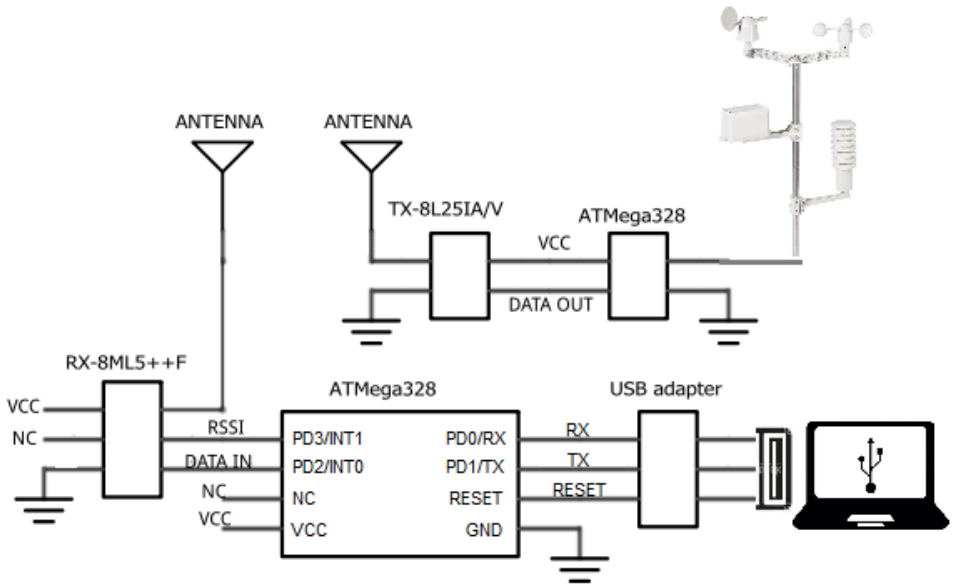

Fig. 1. Simplified embedded weather station

The external interrupt procedure began by reading the status of the wireless modem data lines and identifying which slope caused the interrupt: ascending or descending (Fig. 3). A descending slope ended in a positive pulse and then the timer interrupt L count was checked, which as a multiple of the 100 us interval it

enabled the pulse duration to be determined. An L value between 4 and 6 was considered a logical 1, and an L value between 14 and 16 was considered a logical 0. Otherwise, all counters were reset and another frame was read. Once a logical 0 or 1 bit was identified, the BIT flag was set. The pause duration between pulses was not analysed as the algorithm already had sufficient protection.

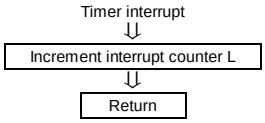

Fig. 2. Microcontroller timer interrupt algorithm

The main program waited for the BIT flag (Fig. 4). When it received the flag, it checked if a preamble had been received. If not, it inserted the bit into the preamble using the bit rotation method and compared it with the preamble reference. If unsuccessful, it waited for another bit. When a preamble was identified, subsequent bits were inserted by bit rotation into the corresponding frame bytes. Bits were counted from 0 to 7, and frame bytes from 0 to 8. Once all frame bytes were filled, the CRC checksum was checked using a polynomial division-based algorithm identical to the one used in the transmitter.

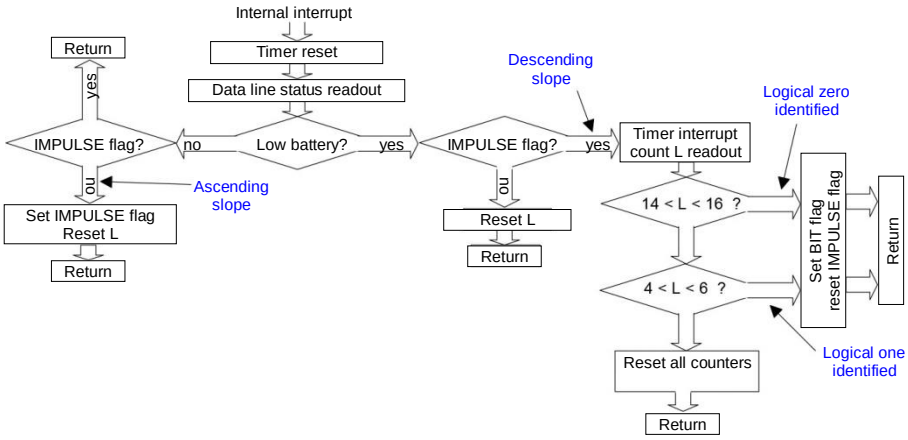

Fig. 3. Microcontroller external interrupt algorithm

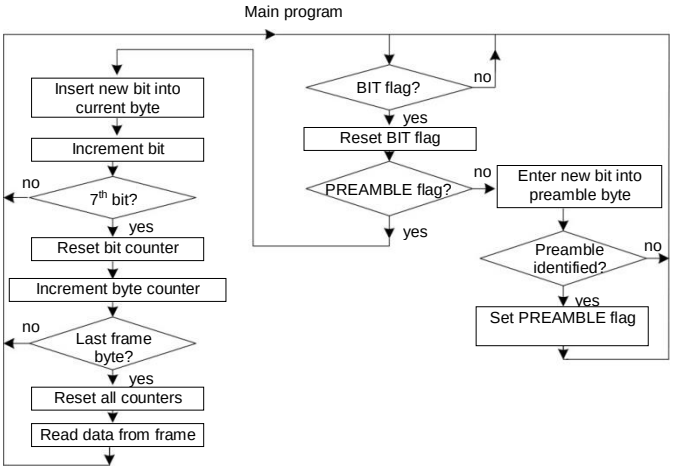

Fig. 4. Microcontroller main program algorithm

The microcontroller prepared the received weather data ready to be sent to a PC over a serial port. To this end, the data were separated with tabulation characters to maintain compatibility with the requirements of the LabVIEW *Write Delimited Spreadsheet.vi* instrument, which saves data in a *.lvm text file and identifies individual data fields by their separating tabulation characters.

3. SERVER APPLICATION FOR DATA ACQUISITION AND COLLECTION

In the DataSocket library [Halvorsen 2015], the LabVIEW environment offers a range of virtual instruments for client-server TCP communication, which were used to design the *OPC_SERVER.vi* application for data acquisition and collection. The application consists of two while() loops known as *"Serial communication with microcontroller"* and *"DataSocket Write Loop"*. These function independently and continuously until the application is stopped or a system or communication error occurs in either of them (Fig. 5).

The while() loop *"Serial communication with microcontroller"* is responsible for establishing serial communication with the microcontroller and for receiving sensor data. Virtual instruments from the VISA library were used for this purpose, including: *VISA Configure Serial Port, VISA Write, VISA Read and VISA Close*, which in the LabVIEW environment support serial communication.

Standard parameters were used for serial transmission with the microcontroller: 9600bps, 8n1 (Fig. 5). An algorithm was selected, under which the *VISA Write.vi* instrument cyclically sends the "s" character with a defined time raster, e.g. 1 minute, and in response the microcontroller sends a sensor data

package. The data package is received by the *VISA Read.vi* instrument as a string, then the decimal stop character, used in the C language for microcontrollers, is replaced with the decimal comma character used in the LabVIEW environment. The character replacement is done in the *Search and Replace String.vi* instrument.

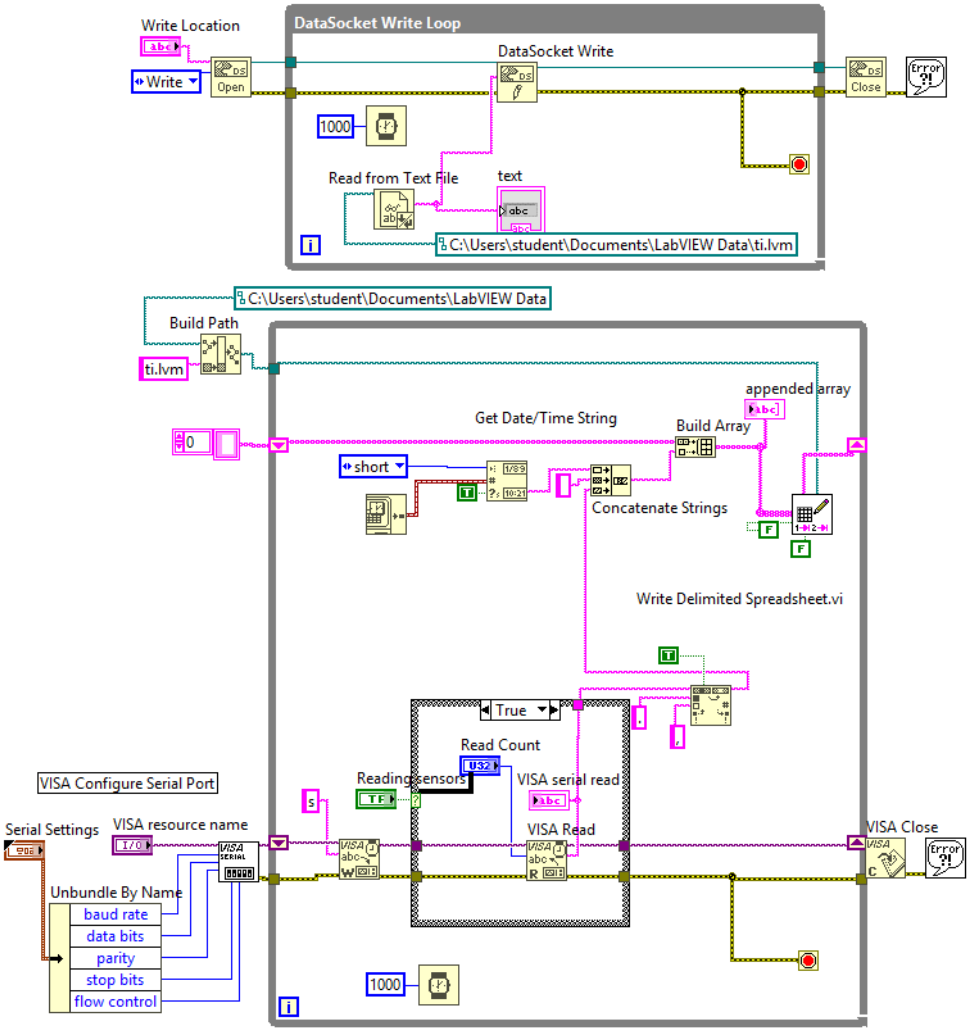

Fig. 5. LabVIEW OPC_SERVER.vi virtual instrument for acquisition and collection of measurement data

Sensor data are combined (*Concatenate String.vi*) with the system clock time to build a two-dimensional array where consecutive lines are added to a *.lvm text file by the *Write Delimited Spreadsheet.vi* instrument each time new data are

received over the serial connection. The while() loop called *"DataSocket Write Loop"* associates the *.lvm data file with DataSocket server resources.

The DataSocket application (…\National Instruments\DataSocket\cwdss.exe) should be enabled in LabVIEW services and DSTP (DataSocket Transport Protocol) selected, indicating the measurement data collected locally (dstp://localhost/wave) – they have the default name "wave". An active DataSocket server acquires new versions of the indicated weather sensor data file cyclically as long as the program is running, using the *DataSocket Write.vi* virtual instrument. These data are then shared in the local IP network to connected DataSocket service clients.

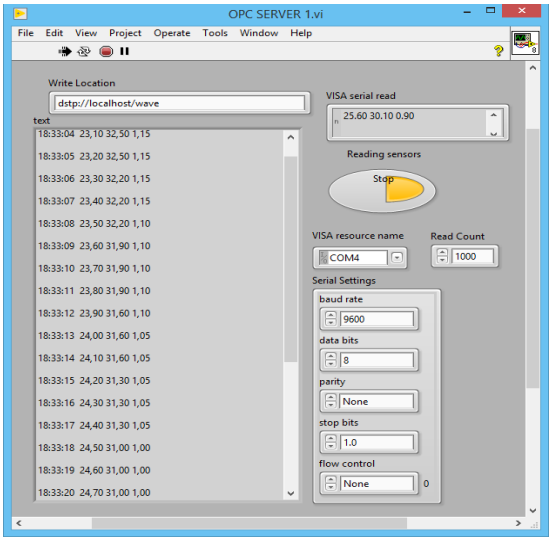

Fig. 6. User panel of the OPC_SERVER.vi instrument for acquisition and collection of measurement data

Thanks to the two independent while() loops in the *OPC_SERVER.vi*, it is possible to concurrently execute serial communication with the downloaded data saved to a file, and to update the data file in the DataSocket network server resources. The data file format adopted facilitates their visualisation in charts vs time on the client's side (Fig. 6).

4. CLIENT APPLICATION FOR DATA VISUALISATION

The DataSocket client application called *OPC_CLIENT.v* is run on another computer in the local IP network (Fig. 7). The *Open File.vi* instrument opens the *.lvm file remotely over the network on the DataSocket server in read-only mode.

The file address is preceded by the IP number of the DataSocket server, and file transfer is effected using the DSTP protocol (similar to FTP). The *SpreadSheet String to Array.vi* instrument inserts the string characters read in a two-dimensional array. The first line, which can contain incomplete data, and the left column are deleted from the array due to the adopted microcontroller transmission method, which begins transmissions starting with the tabulation character, creating an empty left column in this manner. Following this operation, the leftmost column of the resulting array contains the time in hours, minutes and seconds, while subsequent columns transfer the temperature, humidity and wind speed values.

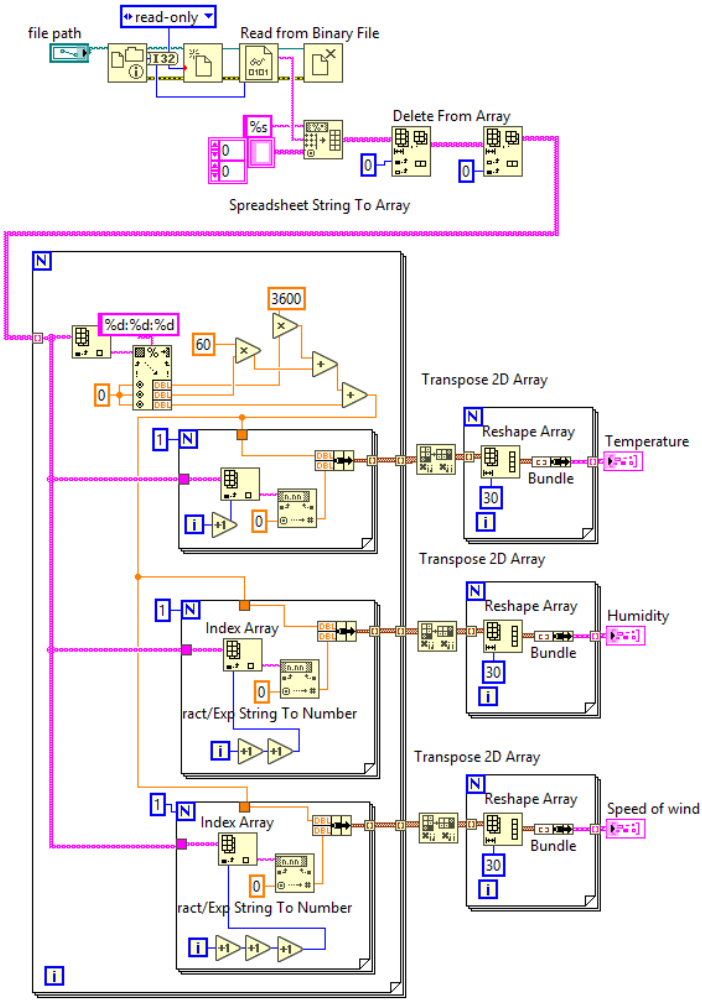

Fig. 7. LabVIEW OPC_CLIENT.vi virtual instrument for visualisation of measurement data

Reading each column separately is possible when the table is transformed into an indexed one in the *Index Array.vi* instrument. Next, the hour, minute and second values are taken from the first column as numbers and translated to the current number of seconds, which is paired with the data from the second, third and fourth columns to draw three charts. Following the transposition, the coordinates of the subsequent charts are fed to an indicator which creates a chart of 30 points which correspond to the weather data collected during the last hour.

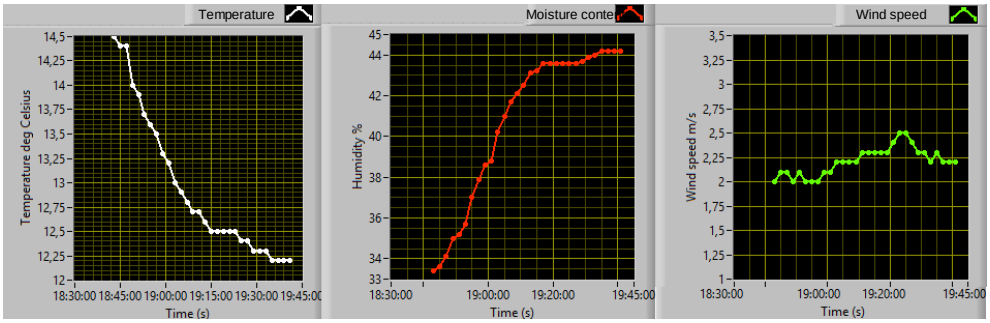

Fig. 8. Measurement data visualisation in the DataSocket network client application

5. SUMMARY

The project constitutes a practical confirmation of the ability of an embedded microprocessor system to work with a DataSocket client-server network based on TCP/IP and FTP protocols. Data collected on a DataSocket server remain available for further analysis, and DataSocket clients in the local IP network are able to view charts drawn on the basis of current and historical weather data values. The solution presented here can also be used for other applications using distributed measurement systems.

REFERENCES

Halvorsen, H., 2015, *DataSocket Simplifies Live Data Transfer for LabVIEW*, http://www.ni.com/pdf/datasocket/us/datasocketarticle.pdf.

Khera, N., Balguvhar, S., 2013, *Development of a Simple Microcontroller Based Real-Time Data Acquisition and Alert System in LabVIEW Environment*, Proceedings of International Conference on Reliability, Infocom Technologies and Optimization (ICRITO'2013), Noida, India.

Pałczyńska, B., Rabczuk, D., 2018, *Low-cost Embedded System for Environmental Monitoring Over the Ethernet with LabVIEW User Interface*, Proceedings of 18th Annual International Conference on Environmental and Electrical Engineering, 12–15 June, Palermo, Italy.

Popa, M., Iapa, C., 2011, *Embedded Weather Station with Remote Wireless Control*, 19th Telecommunication Forum (TELFOR) Proceedings of Papers, 22–24 November, Belgrade, Serbia.

Shaout, A., Li, Y., Zhou, M., Awad, S., 2014, *Low Cost Embedded Weather Station with Intelligent System*, 10th International Computer Engineering Conference ICENCO, Giza, Egypt.

Szlas, F., 2018, *Projekt i wykonanie autonomicznej mikroprocesorowej stacji pogodowej*, praca dyplomowa inżynierska, Akademia Morska w Gdyni, Wydział Elektryczny, Gdynia.

Yiwei, W., Mengling, W., Chun, T., Tianhe, M., 2017, *Development of Remote Data Acquisition System Based on OPC for Brake Test Bench*, Journal of Physics, The 2nd Annual International Conference on Information System and Artificial Intelligence (ISAI), Tianjin, China.