# AN IMPLEMENTATION OF A CONFIGURABLE SERIAL-TO-ETHERNET CONVERTER USING LWIP

**Jungho Moon**[1,*], **Myunggon Yoon**[2]
[1]*Department of Electrical Engineering, Gangneung-Wonju National University, Gangneung, Gangwondo, South Korea*
[2] *Department of Precision Mechanical Engineering, Gangneung-Wonju National University, Wonju, Gangwondo, South Korea*
[*]For correspondence; Tel. + (82) 336402427, E-mail: itsmoon@gwnu.ac.kr

***ABSTRACT:*** *This paper gives a brief introduction to an implementation of an embedded device for converting data from an asynchronous serial communication port to TCP/IP or UDP packets and vice versa using an ARM-based 32-bit microcontroller. The operating parameters of the converting device can be modified through the Ethernet interface of the device using a dedicated configuration program running on a PC. The configuration program was written in Python. The size of serial data to be transmitted as a single packet and the minimum time interval to distinguish different groups of serial data are configurable. The feature facilitates the transmission of received serial data when the received data are organized into groups and each group is to be transmitted as a packet over the Ethernet connections. The converting device supports 3 asynchronous serial communication protocols, RS-232, RS-422, and RS-485 and supports 3 network modes, TCP/IP server, TCP/IP client, and UDP client. The TCP/IP and UDP protocols were implemented on the 32-bit microcontroller based upon lwIP (A lightweight TCP/IP), an open source TCP/IP stack. In the implementation, some features related to Nagle's algorithm, which are usually included in the TCP/IP stack, are disabled for the rapid transmission of data. The converting device can be utilized as a seamless bridge between an existing serial port and a TCP/IP network port without the need for changing existing hardware or protocols.*

**Keywords:** Serial, Ethernet, converter, lwIP, embedded systems

## 1. INTRODUCTION

Communications over the Ethernet connections are frequently used these days in a variety of devices ranging from small embedded devices to personal computers. Despite the various advantages of Ethernet communication, plenty of legacy devices that only have serial communication interfaces are still in use in many fields for the purpose of device configuration and data exchange.

In industrial environments such as factories, it is common to construct a network composed of a plurality of apparatuses and a data logger, where the data logger collects data from the apparatuses on a regular basis. Data loggers for this purpose commonly have Ethernet communication interfaces but not all apparatuses do. Some apparatuses may have only traditional serial communication interfaces. It is not easy to construct such a network using such legacy apparatuses. The traditional communication interface of the legacy apparatuses needs to be upgraded, which may cause both considerable cost and time. A converting device that can convert TCP/IP or UDP packets to asynchronous serial data and vice versa can help solve the problem without the need for modifying the existing legacy apparatuses. The serial-to-Ethernet converter can simply bridge seamlessly the existing the Ethernet interface of the data logger and the asynchronous serial ports of the legacy apparatuses.

This paper gives a brief introduction to an implementation of such a serial-to-Ethernet converter (hereinafter referred to as the converter) using a 32-bit microcontroller STM32F107 and the lwIP (A lightweight TCP/IP). The STM32F107xx series are connectivity line microcontrollers equipped with an Ethernet peripheral that supports both the media independent interface (MII) and the reduced media independent interface (RMII) [1]. The lwIP is an open source TCP/IP stack designed for embedded systems having limited resources [2]. Figure (1) shows a picture of the developed converter.

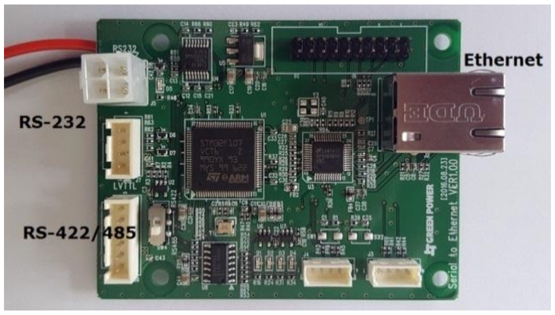


**Fig (1) A picture of the Serial-to-Ethernet converter**

The developed converter supports three asynchronous serial communication protocols, RS-232, RS-422, and RS-485 and supports three different network modes, TCP/IP server, TCP/IP client, and UDP client. Users can configure various operating parameters of the converter through its Ethernet interface. For configuring the converter, we developed a dedicated program running on a PC. It is assumed that the IP address of the converter is not known to the user beforehand. Nonetheless, the configuration program can communicate with the converter and allows the user to read and to modify its operating parameters.

The open source TCP/IP stack, lwIP, supports well-known protocols such as IP, ICMP, UDP, TCP, DHCP, and ARP required for the implementation of the converter. It provides both a raw API and a high-level sequential API that requires a real-time operation system [3]. The raw API is used for developing callback-based applications. Compared to the high-level sequential API, the raw API adds some complexity for application development but provides the best performance and code size [3]. We adopted the raw API for better execution speed and less memory usage. Some features of TCP/IP were disabled for rapid transmission of data at the cost of transmission efficiency.

## 2. CONVERTER'S OPERATING PARAMETERS

A user needs to configure the operating parameters of the converter before using it. The user-configurable parameters include the following:

- Converter's IP address, subnet mask, and gateway's IP address
- Operating mode: TCP/IP server, TCP/IP client, and UDP client
- Port number to which other TCP/IP clients connect when the converter acts as a TCP/IP server
- Server's IP address and port number when the converter acts as a TCP/IP client
- Server's IP address and port number in when the converter acts as a UDP client
- Serial communication parameters: Baud rate, data bits, stop bits, and parity
- Size of serial data to send as a single packet over the Ethernet connection (SOD)
- Timer's fire interval for determining the boundary between two groups of serial data (TFI)

There is no option about the selection of an asynchronous serial communication protocol. All of the three asynchronous serial communication protocols are supported simultaneously but only one of them can be used at a time to avoid conflicts. Users can simply insert a connector to one of the serial communication ports that they want to use.

Receiving a data packet from the Ethernet interface, the converter sends the payload contained in the received packet to the serial communication interface immediately. On the other hand, the asynchronous serial communication is byte-wise and a received byte is not immediately sent to the Ethernet interface. The received serial data are organized into groups and then one group is transmitted to the Ethernet interface as one packet. The two parameters named SOD and TFI specify how to organize and transmit the received serial data to the Ethernet interface.

The TFI indicates the minimum time gap between two consecutive bytes before an internal timer expires as shown in figure (2). If the time interval between two consecutive bytes is less than the TFI value, the two bytes are considered to belong to the same group and stored sequentially in an internal buffer of the microcontroller. If the time interval exceeds the TFI value, the internal timer expires, which indicates the reception of a group of data has been completed and so triggers the transmission of the stored data.

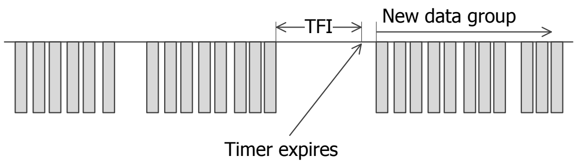
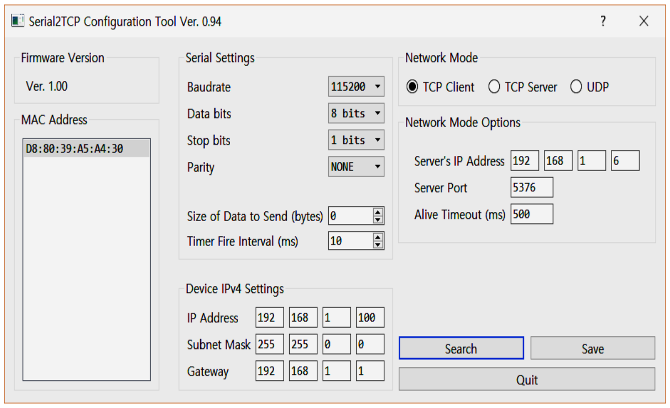


**Fig (2) The relation between the TFI and timer**

If the timer expires, the converter compares the size of the stored data and the SOD value. The data stored in the buffer are transmitted only when the two values are identical. If the two values do not match, the stored data are discarded and the buffer is cleared. If the SOD is set to 0, the converter sends the stored data without regard to its size. If the TFI is set to 0 and the SOD is set to a positive value, the timer is not used and the stored data are transmitted as soon as its size reaches the SOD value. It is not allowed to set both SOD and TFI to zero.

Setting both TFI and SOD to positive values is useful when the serial data are organized into groups and the time interval between two consecutive groups is fixed and known. This is usually the case in real applications in which the serial data are packetized. For example, if the serial data are grouped into packets, the size of each packet being fixed to 100 bytes and the time interval between two consecutive packets being fixed to 50 ms, the SOD can be set to 100 and the TFI can be set a value less than 50 ms. In some applications, the packets may have variable lengths, in which case the SOD can be set 0. Then only the time interval between two consecutive packets is considered.

For setting/modifying these parameters, we developed a dedicated configuration program running on a PC. Figure (3) shows a screen snapshot of the configuration program. The configuration program communicates with converters on the same subnet as the PC on which it is running. It displays the operation parameters of all the converters existing on the subnet and allows the user to modify the parameters of a user-chosen converter. The modified values are transmitted to the user-chosen converter and stored in the internal flash memory of the converter. The configuration program was written in Python using a graphic library called PyQT [4-7]. A Python script is a program executed by the Python interpreter, but there are ways to compile Python scripts into a standalone executable. The configuration program was also built into a standalone application.



Firmware Version: Ver. 1.00
MAC Address: D8:80:39:A5:A4:30

Serial Settings
Baudrate: 115200
Data bits: 8 bits
Stop bits: 1 bits
Parity: NONE
Size of Data to Send (bytes): 0
Timer Fire Interval (ms): 10

Device IPv4 Settings
IP Address: 192 168 1 100
Subnet Mask: 255 255 0 0
Gateway: 192 168 1 1

Network Mode: [✓ TCP Client] TCP Server UDP

Network Mode Options
Server's IP Address: 192 168 1 6
Server Port: 5376
Alive Timeout (ms): 500

**Fig (3) A screen snapshot of the configuration program**

Though the configuration is conducted through the Ethernet connection, the user is not required to know the IP address of the converter. The IP address of the converter is assumed to be not known to the user beforehand. Under this condition, the only method that enables communication between the configuration program running on a PC and the converters on the same subnet as the PC is to exchange UDP broadcast messages. As a result, both the configuration program and the converters use UDP broadcasts for the exchange of requests and responses.

If the user clicks the Search button shown in figure (3), the program sends a UDP broadcast message on a predefined port to discover the converters existing on the network. The broadcast message, which has a predefined format, is received by all devices listening on the port but other devices do not understand it. All the converters always listen on the port and can respond to the broadcast message. The response of each converter includes its firmware version, its MAC address, and aforementioned operating parameters. Receiving the responses from all the converters existing on the network, the configuration program displays the information contained therein, as shown in figure (3). The configuration program can display the operating parameters of all the converters but the parameters of one converter at a time. Since each converter has a unique MAC address, the user can identify the parameters of a specific converter by choosing its MAC address listed on the left side of the configuration program shown in figure (3).

After the user chooses a specific MAC address, the user can modify the operating parameters of the converter associated with the chosen MAC address. If the user clicks the Save button after setting all the necessary parameters, the configuration program sends a UDP broadcast message on the predefined port, the message containing the MAC address of the target converter and all the operating parameters. The broadcast message is received by all the converters on the same subnet. The converters discard the broadcast message unless the MAC address contained therein matches its own MAC address. As a result, only the converter that has the same MAC address accepts the parameters contained in the broadcast message and writes them in its internal flash memory. After completing the writing operation, the convert reboots.

## 3. THE ALGORITHM FOR HANDLING DATA

The 32-bit microcontroller handles both the transmission and reception of the serial data using interrupts. As soon as a new byte is received by the serial communication interface, an associated receive interrupt handler is executed. Figure (4) shows the flow chart for the data processing of the receive interrupt handler.

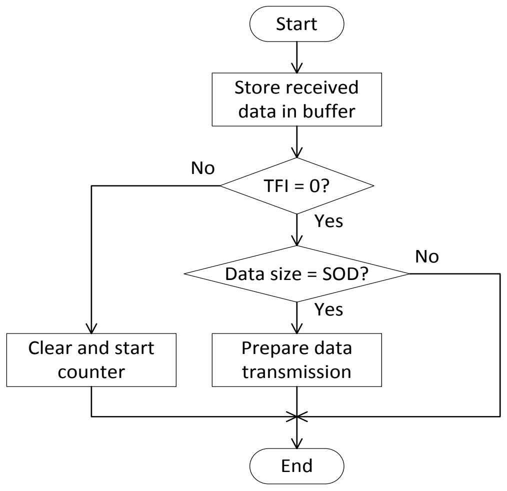


**Fig (4) The flow chart for the processing of the serial data receive interrupt**

Receiving a new byte, the receive interrupt handler stores it in an internal buffer and checks whether the TFI value is zero. If so, the handler compares the size of the stored data to the SOD value. If the two values are identical, the condition for transmitting the serial data over the Ethernet connection is met and the interrupt handler prepares for the transmission of the store data. If the two values do not match, no further action is executed. If the TFI is set to a positive value, the receive interrupt handler clears the timer for measuring the time interval between two consecutive bytes and starts the timer. If the timer fires before a next byte is received, an associated timer interrupt handler is executed. In other words, in the case of a positive TFI value, whether to transmit the stored serial data over the Ethernet connection is determined not by the receive interrupt handler but by the timer interrupt handler.

If the timer expires when the TFI is set to a positive value, it indicates that the reception of a group of data has been completed and data to be received subsequently will belong to a new group. Unless the SOD is set to zero, the timer interrupt handler compares the size of the data that have been stored in the buffer thus far to the SOD value. If the two values match, the timer interrupt handler prepares for the transmission of the stored data. Otherwise, the data in the buffer are simply discarded. If the SOD is set to zero, the timer interrupt handler prepares for the transmission of the stored data with no further comparison.

The receive interrupt handler and the timer interrupt handler only determine whether the condition for transmitting the stored data is satisfied. The actual data transmission is handled in the main routine because it is a time-consuming task and therefore it is not desirable to execute it in the interrupt handlers. The main routine has an infinite loop in which there is a routine to check whether the preparation for the data transmission is finished. If confirmed, several lwIP API functions are called sequentially to transmit the stored data over the Ethernet connection. The API functions to be called depend upon the current network mode of the converter.

New data can be received by the serial interface while the data stored in the buffer are being transmitted over the Ethernet connection, which may result in data corruption or transmission of the wrong data. To prevent such a problem, we need two separate buffers, one for storing new data and the other for storing the old data to transmit. The converter manages two pointers, one pointing to the buffer for storing new data and the other pointing to the buffer storing the data to transmit. The former is used by the receive interrupt handler and the latter is used by the routine to transmit data. Figure (5) shows the mechanism for maintaining the two buffers.

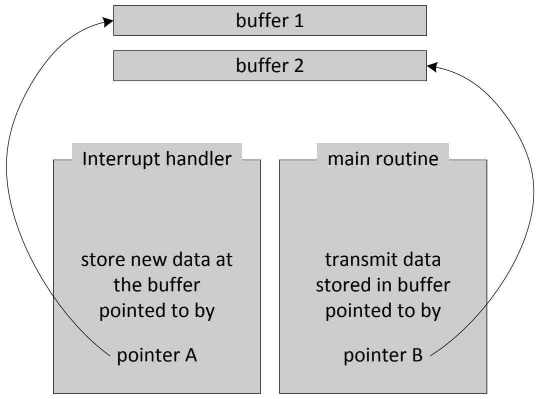

buffer 1
buffer 2
Interrupt handler
store new data at the buffer pointed to by
pointer A
main routine
transmit data stored in buffer pointed to by
pointer B

**Fig (5) The management of dual buffers**

In the figure, the point A and point B point to buffer 1 and buffer 2, respectively. The receive interrupt handler stores incoming data in the buffer pointed to by the pointer A and the main routine transmits data stored in the buffer pointed to by the pointer B. Under this condition, incoming data are stored in the buffer 1 and the data stored in the buffer 2 are transmitted to the Ethernet interface. When the receive interrupt handler concludes that the condition for transmitting the data stored in the buffer pointed to by the pointer A is met, it prepares for the transmission of the stored data. After completing the preparation operation, the receive interrupt handler immediately switches the contents of the two pointers. From this point on, the incoming data will be stored in the buffer 2 by the receive interrupt handler and the data stored in the buffer 1 will be transmitted by the main routine, which can effectively prevent the data to be sent from being corrupted by the newly received data. The timer interrupt handler does the same switching operation as the receive interrupt handler does.

Unlike the serial data processed by interrupt handlers, data received from the Ethernet interface are processed by lwIP callback functions. All the necessary callback functions are registered during the initialization of the lwIP. The TCP or UDP operations are based upon continuous software polling to check if a new packet is received [3]. Figure (6) explains the operation model of the lwIP in the case where the raw API is used. The continuous polling operation is executed within an infinite loop in the main routine and therefore it is desirable that the infinite loop should not contain time-consuming functions like delay functions. In the flow chart shown in figure (6), all the tasks except for the last step are handled by the lwIP stack and only the last step needs to be handled by the application.

Exceptions and errors are also handled by associated callback functions. For example, if the number of SYN retries reaches a predefined maximum value when trying to open a TCP connection, the application is notified of the error via an associated callback function, i.e., the associated callback function is called automatically. It is not the lwIP but the application that is responsible for handling the error appropriately via the callback function.

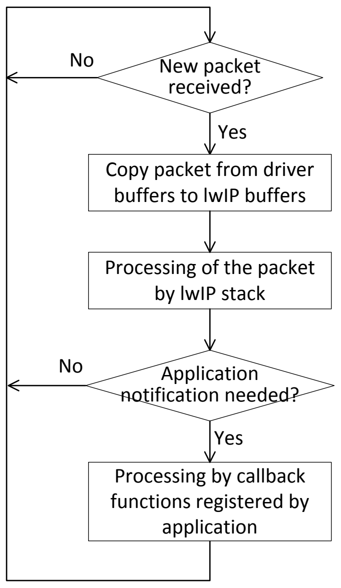

New packet received?
No
Yes
Copy packet from driver buffers to lwIP buffers
Processing of the packet by lwIP stack
Application notification needed?
No
Yes
Processing by callback functions registered by application

**Fig (6) The operation model of the lwIP when raw API is used**

The converter may encounter various kinds of exceptions and errors during its operation, but must work fine without human intervention. The UDP is connectionless and works based on the principle “send and forget.” When the converter works as a UDP client, it simply transmits received serial data; therefore the converter in this mode is not likely to encounter exceptions. In contract, the TCP is a connection-oriented protocol that provides reliable data delivery services. When the converter is configured as a TCP server or client, the advantage of the TCP may create unexpected problems since the converter is a small embedded system with a limited memory.

A TCP connection may get broken for some reasons while the converter transmits data packets to the Ethernet interface. The TCP/IP stack has no means to detect the broken connection immediately. In this case, the converter does not receive acknowledgements and cannot clear the TCP transmit buffer. The converter keeps retrying the transmission of the unacknowledged data packets until a predefined maximum number of retransmissions is reached. In addition, the apparatus connected to the serial port of the converter does not know the existence of a TCP connection. It can transmit data to the converter even while the converter does not recognize that the TCP connection is broken. If this happens, the transmit buffer of the converter keeps being consumed and is not freed until the predefined maximum number of retransmissions is reached. This would not cause a critical problem in systems with sufficient memory. The converter, however, has a limited memory space, and therefore, will be short of memory soon. Such situations are of primary concern in the implementation of the converter.

As mentioned above, the data received from the Ethernet interface are processed by the lwIP callback functions. The converter needs to manage two sets of callback functions. The first set is UDP-related callback functions and this is irrespective of the network mode of the converter. These callback functions are required for configuring the converter. Since the dedicated configuration program communicates with the converter using UDP broadcasts, the converter need to maintain this set of callback functions. Every task required for configuring the converter, such as parsing the received UDP broadcast messages, writing the received configuration data to the internal flash memory of the microcontroller, and responding to the query from the configuration program, is performed by these callback functions. The second set is callback functions related to the current network mode of the converter. Since the converter has three network modes, the converter needs three different groups of callback functions. It should be pointed out, however, that only the group of callback functions pertaining to the current network mode of the converter is registered during the initialization of the converter because the other two groups of callback functions are not used in the current mode. This is why the converter reboots after the configuration program modifies its network mode. Registering all the three groups of callback functions leads to the waste of memory, and therefore, should be avoided.

Another issue to consider is the so-called Nagle’s algorithm for improving the efficiency of TCP/IP networks by reducing the number of packets that need to be transmitted over the network. Nagle's algorithm works by combining a number of small outgoing messages, and sending them all at once. Specifically, as long as there is a sent packet for which the sender has received no acknowledgment, the sender should keep buffering its output until it has a full packet's worth of output, so that output can be sent all at once [8]. The lwIP also includes Nagle’s algorithm in the implementation of TCP.

In most cases, Nagle’s algorithm does not yield problems for the operation of the converter when it is configured as a TCP/IP server or client. It was occasionally observed, however, that the converter tried to combine two or three groups of serial data into one packet especially when the size of serial data to be sent as one packet is very small and the time interval between two consecutive data groups is relatively short. This is the effect of Nagle’s algorithm. The threshold value for the time interval that causes such a phenomenon may depend on applications. The converter should work in such a way that it always transmits a group of serial data as a single packet. To this end, Nagle’s algorithm included in the lwIP needs to be disabled for normal operation of the converter.

## 4. CONCLUSIONS

The paper introduced an implementation of a configurable Serial-to-Ethernet converter using the lwIP stack and a 32-bit microcontroller. The developed converter supports three asynchronous serial communication protocols and three network modes. We adopted the raw API of the lwIP for the implementation of the converter for the purpose of achieving better execution speed with less memory footprint. The parameters required for the operation of the converter were explained and the algorithm for the processing of the received serial data was described in detail. Also, the way that the dedicated configuration program communicates with the converter using UDP broadcast was explained. Although the core functions of the lwIP do not need modification, Nagle’s algorithm, which is included in the lwIP for improving the efficiency of the TCP/IP networks, needs to be disabled for the rapid transmission of data. It is especially inevitable when the size of a packet is very small the time interval between two consecutive packets is short. The developed converter can work as a seamless bridge between an existing serial port and a TCP/IP network port without the need for changing existing hardware or protocols.

## 5. REFERENCES

[1] *STM32F107xxx Reference Manual*, STMicroelectronics (2014).
[2] “lwIP Wiki”, Available at: http://lwip.wikia.com/wiki/LwIP_Wiki. [Accessed Sep. 20, 2016].
[3] *Developing Applications on STM32Cube with lwIP TCP/IP Stack*, STMicroelectronics (2015).
[4] Lubanovic, B., *Introducing Python: Modern Computing in Simple Packages*, O’Reilly (2014).
[5] Lutz, M., *Learning Python 5/e*, O’Reilly (2013).
[6] Summerfield, M., *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*, Prentice Hall (2015).
[7] Harwani, B. M., *Introduction to Python Programming and Developing GUI Applications with PyQT*, Cengage Learning PTR (2011).
[8] “Nagle’s algorithm”, Available at: https://en.wikipedia.org/wiki/Nagle%27s_algorithm, [Accessed Sep. 10, 2016].

*For correspondence; Tel. + (82) 336402427, E-mail:itsmoon@gwnu.ac.kr