World Academy of Science, Engineering and Technology
International Journal of Information and Communication Engineering
Vol:9, No:12, 2015

# Development of a Serial Signal Monitoring Program for Educational Purposes

Jungho Moon, Lae-Jeong Park

***Abstract*—**This paper introduces a signal monitoring program developed with a view to helping electrical engineering students get familiar with sensors with digital output. Because the output of digital sensors cannot be simply monitored by a measuring instrument such as an oscilloscope, students tend to have a hard time dealing with digital sensors. The monitoring program runs on a PC and communicates with an MCU that reads the output of digital sensors via an asynchronous communication interface. Receiving the sensor data from the MCU, the monitoring program shows time and/or frequency domain plots of the data in real time. In addition, the monitoring program provides a serial terminal that enables the user to exchange text information with the MCU while the received data is plotted. The user can easily observe the output of digital sensors and configure the digital sensors in real time, which helps students who do not have enough experiences with digital sensors. Though the monitoring program was programmed in the Matlab programming language, it runs without the Matlab since it was compiled as a standalone executable.

***Keywords*—**Digital sensor, MATLAB, MCU, signal monitoring program.

## I. Introduction

INCREASINGLY, many sensors come with a digital output interface such as I$^2$C or SPI. Unlike analog sensors, digital sensors output data in 1's and 0's that can be directly read by an MCU without the need for analog-to-digital conversions. The feature makes hardware design simpler and more efficient. On the other hand, engineers who do not have enough experiences with digital sensors may have a difficulty in visualizing sensor outputs since digital sensor data cannot be simply monitored by an instrument such as an oscilloscope. The sensor outputs can only be viewed after being read by an MCU. If the system in which the MCU is incorporated has a display device, the sensor data could be plotted on the device. This is, however, not the case in most embedded systems.

Almost all commercial MCUs are equipped with at least one UART (Universal Asynchronous Receiver/Transmitter), which facilitates asynchronous serial communication [1]–[3]. Once an asynchronous serial communication channel is established between an MCU and a PC, the MCU can easily transmit sensor data read from one or more digital sensors to the PC. The sensor data received from an MCU are binary values. To plot the data in real time, a special monitoring program running on the PC is required. Whereas serial communication is conducted byte-wise, the digital sensors may have a resolution higher than 8 bits. As a result, the output of a sensor needs to be transmitted after being divided into two bytes and then reassembled into the original value on the PC side. Additionally, the number of sensors that the MCU reads may vary depending on situations and requirements. To deal with the problems, a communication protocol needs to be defined and to be shared between the firmware running on the MCU and the monitoring program running on the PC.

This paper introduces a monitoring program running on a PC developed for plotting digital data transmitted by an MCU in real time. The data is usually, but not necessarily, the output of digital sensors read by the MCU. The received data can be plotted in both the time domain and the frequency domain. The program also includes a serial terminal via which text data can be exchanged between the PC and the MCU while the received data is plotted. In addition, the program allows users to start/stop the transmission of data by issuing control commands to the MCU. The monitoring program is programmed with the C and Matlab programming languages. The developed program allows digital sensor data to be easily observed in graphical forms, thereby helping inexperienced students or engineers to get familiarized with sensors with a digital interface.

## II. Functions of the Program

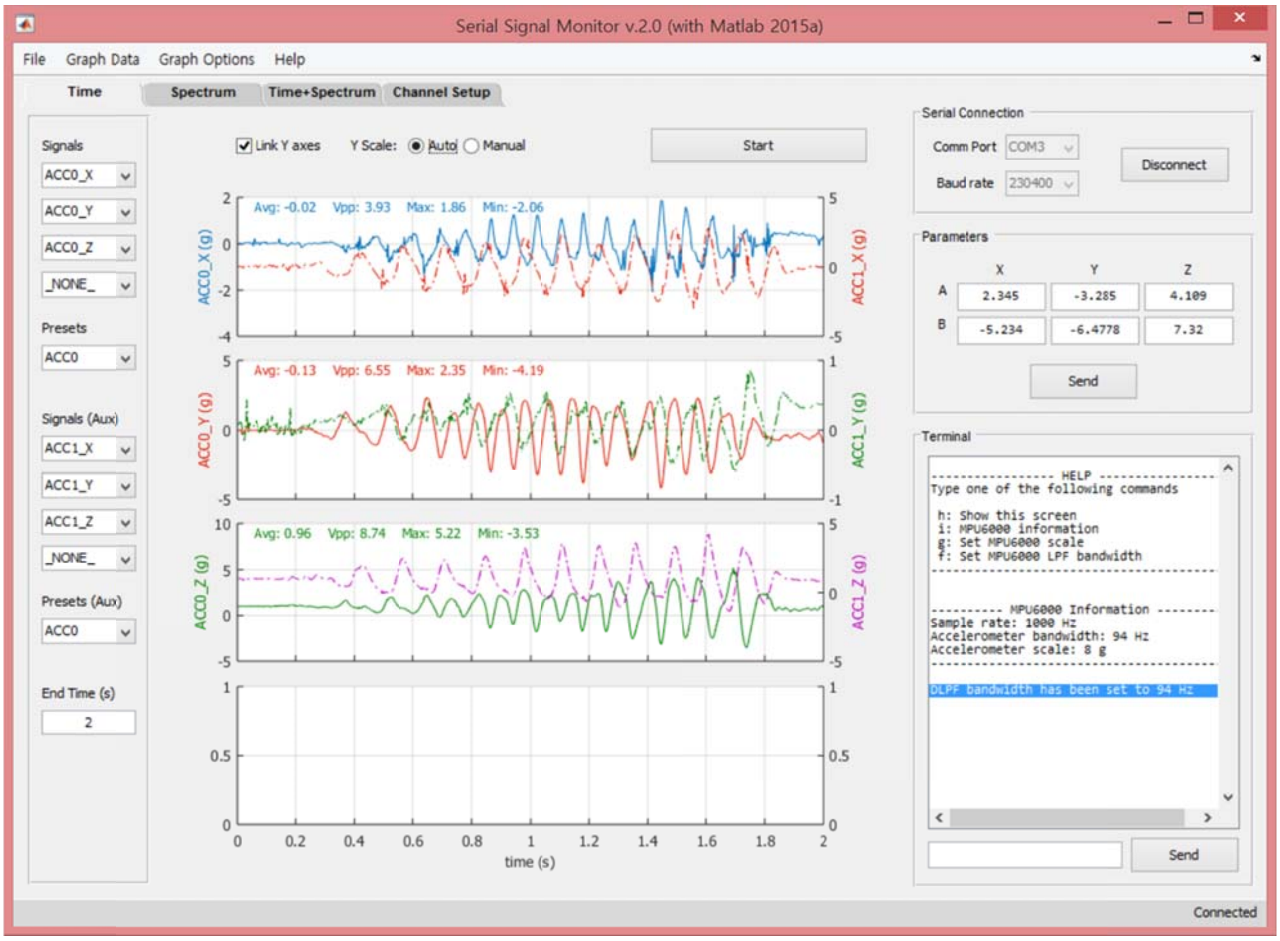
The developed monitoring program includes the following functions:

- Selection of the communication baud rate,
- Selection of the signal sampling rate,
- Individual configuration of channel properties including signal name, signal gain, and signal unit,
- Real-time plot of sensor data of up to 8 channels in both the time and the frequency domains,
- Text terminal for transmitting/receiving text information,
- Transmission/reception of a set of 6 floating-point numbers that could be possibly used to configure parameters of digital sensors or filters implemented on the MCU.

Fig. 1 shows a snapshot of the monitoring program, where the output of two 3-axis accelerometers are plotted in the time domain and some text messages received from an MCU are displayed in the serial terminal. The monitoring program has four tabs: time tab, spectrum tab, time+spectrum tab, and channel setup tab. Each tab except for the channel setup tab has pulldown menus on the left side and plotting areas on the right side. The pulldown menus are for selecting signals to monitor and for setting some plotting parameters like the time-axis limit. The plotting area is positioned in the middle portion of the program and the type of displayed plots varies depending upon the currently chosen tab. The rightmost part of the program has

J. Moon and L.-J. Park are with the Department of Electrical Engineering, Gangneung- Wonju National University, 7 Jukhun-gil, Gangneung 210-702 South Korea, (e-mail: itsmoon@gwnu.ac.kr and ljpark@gwnu.ac.kr).

three panels: a panel for configuring serial communication parameters such as the serial port number and baud rate, a panel for editing and transmitting a set of 6 floating-point parameters, and a text terminal for exchanging text information.



|  | X | Y | Z |
|---|---|---|---|
| A | 2.345 | -3.285 | 4.109 |
| B | -5.234 | -6.4778 | 7.32 |

```
----------------- HELP ----------------
Type one of the following commands

 h: Show this screen
 i: MPU6000 information
 g: Set MPU6000 scale
 f: Set MPU6000 LPF bandwidth
---------------------------------------


---------- MPU6000 Information --------
Sample rate: 1000 Hz
Accelerometer bandwidth: 94 Hz
Accelerometer scale: 8 g
---------------------------------------

DLPF bandwidth has been set to 94 Hz
```

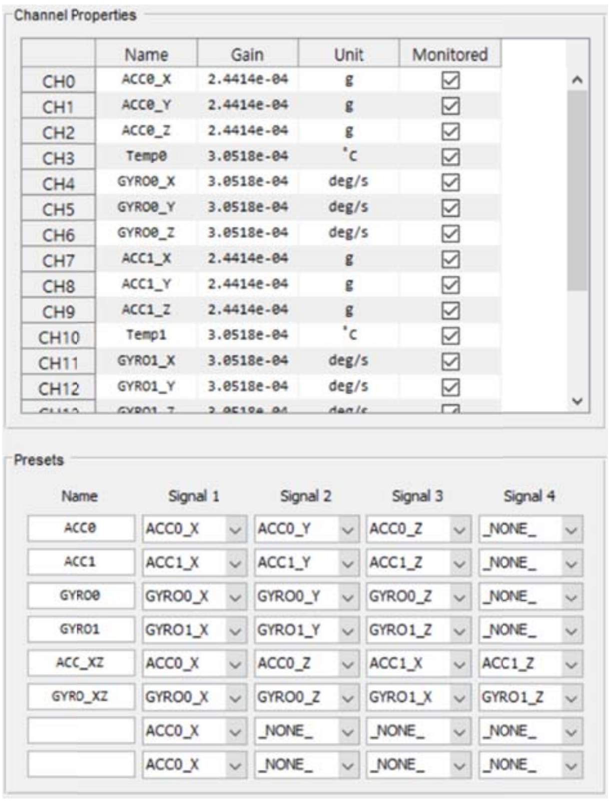
Fig. 1 A snapshot of the monitoring program

Fig. 2 shows the channel setup tab composed of two panels. The upper panel sets channel properties. The signal name, signal gain, and signal unit of each channel can be individually configured. The signal names edited in this panel will appear in the pulldown menus of each graph tab, thereby allowing the user to select signals to monitor using the user-defined signal names. The signal gains are used when converting binary data received from the MCU into corresponding physical quantities such as voltages or deg/s. Signals of up to 8 channels selected among 20 channels can be monitored at the same time. If necessary, the user can prevent some unnecessary signals from appearing in the pulldown menus by unchecking the box in the rightmost side of each column. As mentioned before, the maximum number of signals that can be monitored at the same time is 8. If the number of signals to be monitored exceeds the maximum number, 8 signals need to be chosen one by one, which might be inconvenient. The lower panel of the channel setup tab allows the user to edit presets, each of which defines a group of signals to monitor at the same time. The user can choose signals of up to 4 channels simultaneously simply by selecting a user-defined preset. The configuration and preset data can be stored in a file. The configuration file is reloaded automatically when the monitoring program is loaded.

## III. Implementation of the Program

The length of sensor data is assumed to be 16-bit long but the serial communication is performed byte-wise. In addition, the number of sensor channels varies depending on the user's input. As a result, the sensor data cannot be transmitted to the PC in the form of raw data because the monitoring program cannot distinguish between lower and upper bytes. To deal with this problem, the sensor data are packetized before transmission and the monitoring program and the MCU exchanges packets. A packet is composed of a start of packet delimiter, a payload, and an end of packet delimiter. The size of a payload depends on the types of packets.

The first byte of the payload of a packet is indicative of the type of the packet. The MCU transmits to the PC data packets, text packets, and parameter packets. A data packet carries a set of sensor data read from one or more sensors. A text packet carries text information to be displayed on the text terminal of the monitoring program. A parameter packet carries a set of 6 floating-point parameters stored in the MCU. The program transmits to the MCU command packets, text packets, and parameter packets. A command packet carries one of the following commands: the start of data transmission, the end of data transmission, and querying for the 6 parameters stored in the MCU. A text packet carries text information that the user enters in the terminal. A parameter packet carries a set of 6 floating-point parameters to be stored in the MCU.



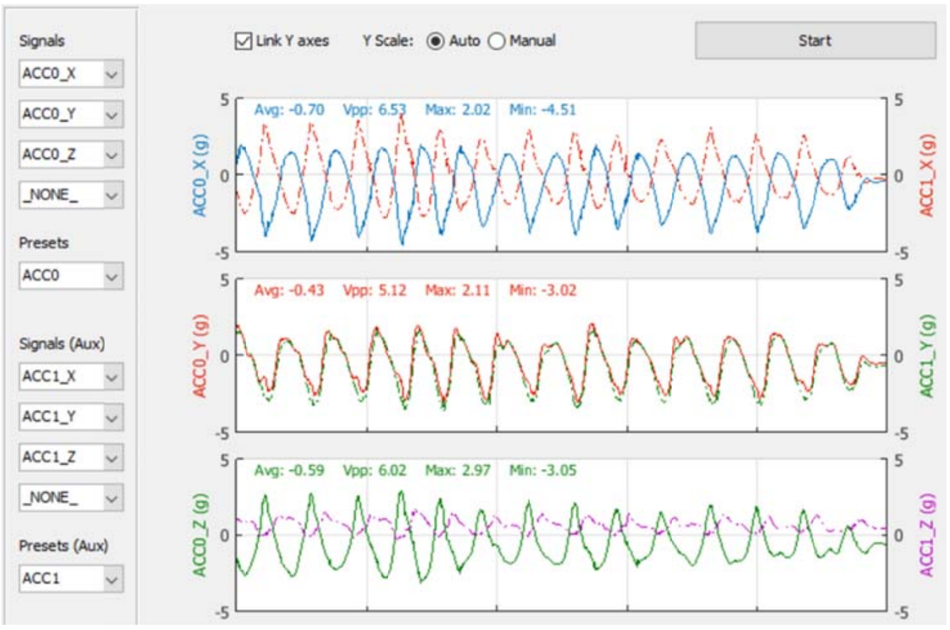
Channel Properties

| | Name | Gain | Unit | Monitored |
|---|---|---|---|---|
| CH0 | ACC0_X | 2.4414e-04 | g | ☑ |
| CH1 | ACC0_Y | 2.4414e-04 | g | ☑ |
| CH2 | ACC0_Z | 2.4414e-04 | g | ☑ |
| CH3 | Temp0 | 3.0518e-04 | 'C | ☑ |
| CH4 | GYRO0_X | 3.0518e-04 | deg/s | ☑ |
| CH5 | GYRO0_Y | 3.0518e-04 | deg/s | ☑ |
| CH6 | GYRO0_Z | 3.0518e-04 | deg/s | ☑ |
| CH7 | ACC1_X | 2.4414e-04 | g | ☑ |
| CH8 | ACC1_Y | 2.4414e-04 | g | ☑ |
| CH9 | ACC1_Z | 2.4414e-04 | g | ☑ |
| CH10 | Temp1 | 3.0518e-04 | 'C | ☑ |
| CH11 | GYRO1_X | 3.0518e-04 | deg/s | ☑ |
| CH12 | GYRO1_Y | 3.0518e-04 | deg/s | ☑ |

Presets

| Name | Signal 1 | Signal 2 | Signal 3 | Signal 4 |
|---|---|---|---|---|
| ACC0 | ACC0_X | ACC0_Y | ACC0_Z | _NONE_ |
| ACC1 | ACC1_X | ACC1_Y | ACC1_Z | _NONE_ |
| GYRO0 | GYRO0_X | GYRO0_Y | GYRO0_Z | _NONE_ |
| GYRO1 | GYRO1_X | GYRO1_Y | GYRO1_Z | _NONE_ |
| ACC_XZ | ACC0_X | ACC0_Z | ACC1_X | ACC1_Z |
| GYRO_XZ | GYRO0_X | GYRO0_Z | GYRO1_X | GYRO1_Z |
| | ACC0_X | _NONE_ | _NONE_ | _NONE_ |
| | ACC0_X | _NONE_ | _NONE_ | _NONE_ |

Fig. 2 The channel setup tab



Signals: ACC0_X, ACC0_Y, ACC0_Z, _NONE_
Presets: ACC0
Signals (Aux): ACC1_X, ACC1_Y, ACC1_Z, _NONE_
Presets (Aux): ACC1
☑ Link Y axes
Y Scale: ● Auto ○ Manual

Avg: -0.70 Vpp: 6.53 Max: 2.02 Min: -4.51
Avg: -0.43 Vpp: 5.12 Max: 2.11 Min: -3.02
Avg: -0.59 Vpp: 6.02 Max: 2.97 Min: -3.05

Fig. 3 Plots shown in the time tab

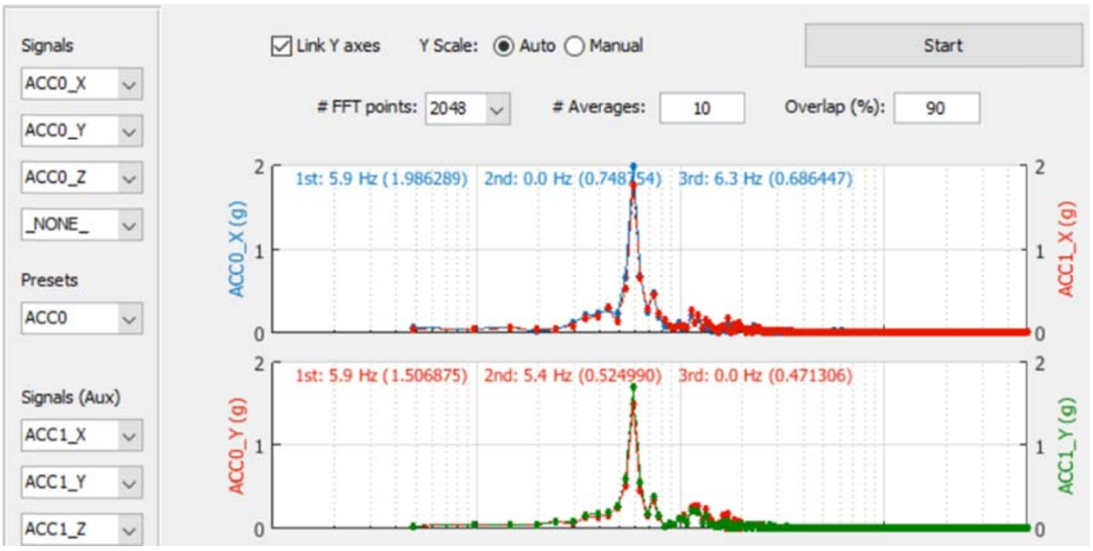


Fig. 4 Plots shown in the spectrum tab

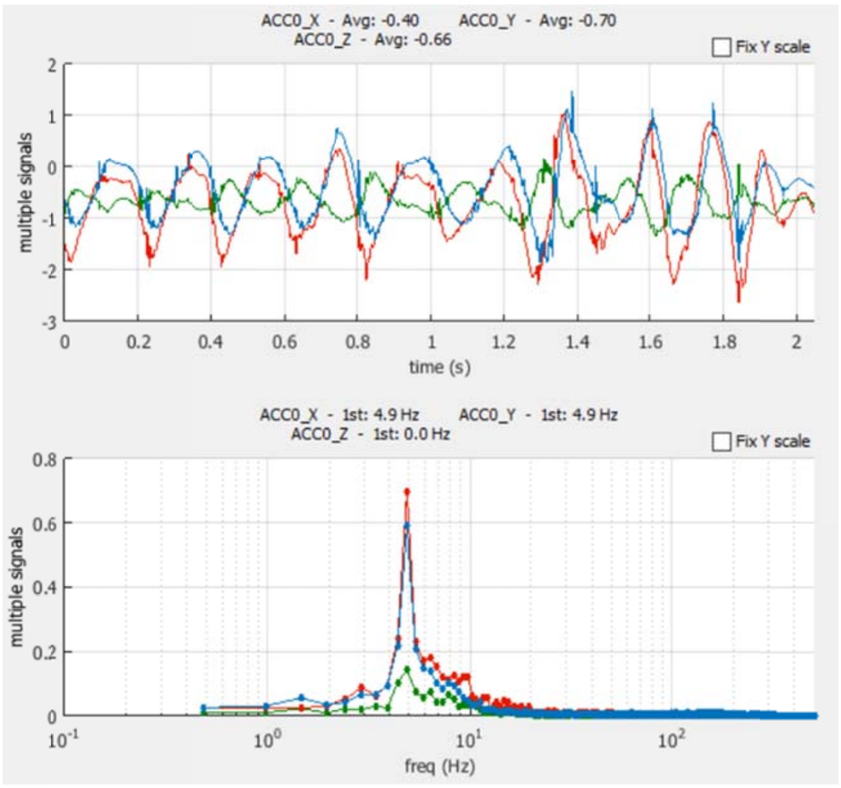
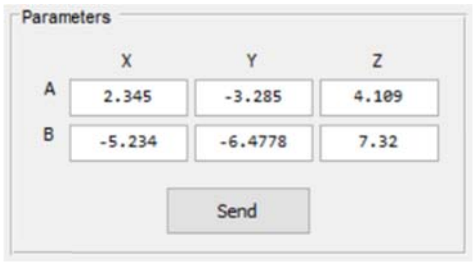


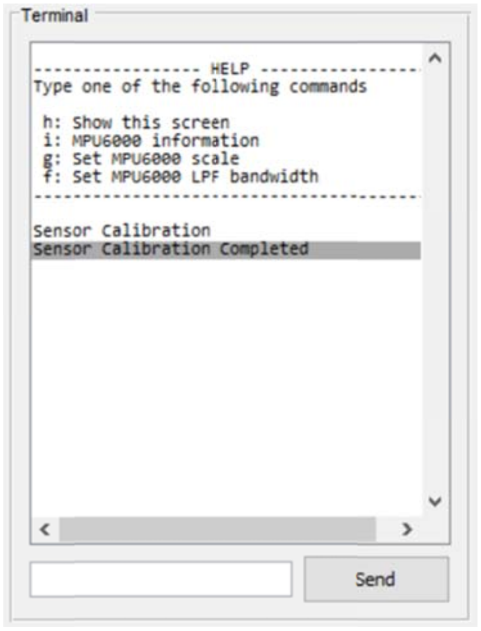
Fig. 5 Plots shown in the time+spectrum tab

The packet delimiters and the packet type information are an inevitable overhead entailed in the packetization. The packet delimiters indicative of the start and end of a packet must be unique in the sense that the delimiters must not occur in the payload of a packet. In the case where one or more delimiters are contained in the payload of a packet, the values should be replaced with different ones, which cause an additional overhead. The probabilistic overhead is 3/256 bytes per a byte of data. To send a data packet containing $n$ bytes of sensor data, the MCU needs to send $\left(1+\frac{3}{256}\right)n+\ 3$ bytes probabilistically. The maximum number of data packets that the MCU can send per second, i.e., the maximum transfer rate, can be calculated based on the information. The maximum data transfer rate is dependent upon the number of sensor channels contained in a data packet and the baud rate. Table I summarizes the maximum data transfer rates that can be achieved at several common baud rates. It should be noted that the maximum data transfer rate is obtained based on the probabilistic calculation; therefore it may not be achieved depending on the contents of payloads. In addition, the data rate when transmitting 2 channels of sensor data in a packet is not twice the data rate when transmitting 4 channels of sensor data in a packet since each packet requires two delimiters and a byte of the type of the packet without regard to the payload size.



|   | X | Y | Z |
|---|---|---|---|
| A | 2.345 | -3.285 | 4.109 |
| B | -5.234 | -6.4778 | 7.32 |

Fig. 6 The panel for editing floating-point parameters



```
---------------- HELP ----------------
Type one of the following commands

 h: Show this screen
 i: MPU6000 information
 g: Set MPU6000 scale
 f: Set MPU6000 LPF bandwidth
--------------------------------------

Sensor Calibration
Sensor Calibration Completed
```

Fig. 7 The text terminal

TABLE I
MAXIMUM DATA TRANSFER RATES

| baud rate | The number of sensor channels contained in a packet | | | |
|---|---|---|---|---|
| | 2 | 4 | 6 | 8 |
| 38400 | 544.9 Hz | 346.1 Hz | 253.6 Hz | 200.1 Hz |
| 57600 | 817.4 Hz | 519.2 Hz | 380.4 Hz | 300.2 Hz |
| 115200 | 1.63 KHz | 1.04 KHz | 760.9 Hz | 600.4 Hz |
| 230400 | 3.27 KHz | 2.08 KHz | 1.52 KHz | 1.20 KHz |
| 250000 | 3.55 KHz | 2.25 KHz | 1.65 KHz | 1.30 KHz |

The main task of the monitoring program is to plot received data in several graphical formats and to perform mathematical operations on the received data set such as FFT (Fast Fourier Transform). The Matlab provides a variety of useful plot functions and allows easy data manipulation and mathematical operations [4]; therefore, the Matlab programming language was chosen to implement the monitoring program. It is, however, inconvenient and inefficient to deal with packet encoding and decoding with the Matlab language because packet handling requires many arithmetic and logical operations on byte arrays. It is more efficient to write the packet handling functions in other programming languages like C that run faster inherently than the Matlab language. As a result, most of the program was written in the Matlab language and only the functions for manipulating packets were written in C. MEX functions written in C can be called by the Matlab code just like other Matlab functions [5]. Despite the fact that the monitoring program was written in the Matlab language, it runs without the Matlab as it was compiled as a standalone application.

## IV. CONCLUSION

This paper gave a brief introduction to a monitoring program developed for real-time plotting of sensor data received from an MCU via an asynchronous serial communication interface. The received data can be plotted in both the time and the frequency domains. Also, the monitoring program can send commands to the MCU to control the data transmission from the MCU and can send a set of 6 floating-point numbers. The developed program can be effectively used in educating students who are not familiar with sensors with a digital interface in that the outputs of digital sensors can be easily visualized using the program.

## REFERENCES

[1] *Atmel ATmega128 Datasheet*. Atmel, 2014.
[2] *STM32F10xxx Reference Manual*. STMicroelectronics, 2014.
[3] *STM32F40xxx Reference Manual*. STMicroelectronics, 2014.
[4] MathWorks, *Matlab Creating Graphical User Interfaces.* MathWoks, 2015.
[5] MathWorks, *Matlab C/C++, Fortran, and Python API Reference.* MathWoks, 2015.