Arduino based Smart Home Automation System

A Simple and Efficient Serial Communication Method

Daniel Chioran[1]
Technical University of Cluj Napoca
Cluj Napoca, Romania

Honoriu Valean[2]
Department of Automation
Technical University of Cluj Napoca
Cluj Napoca, Romania

Abstract—**Around the World massive quantities of energy are consumed in residential buildings leading to a negative impact on the environment. Also, the number of wireless connected devices in use around the World is constantly and rapidly increasing, leading to potential health risks due to over exposer to electromagnetic radiation. An opportunity appears to reduce the energy consumption in residential buildings by introducing smart home automation systems. Multiple such solutions are available in the market with most of them being wireless, so the challenge is to design such systems that would limit the quantity of newly generated electromagnetic radiation. For this we look at several wired, serial communication methods and we successfully test such a method using a simple protocol to exchange data between an Arduino microcontroller board and a Visual C# app running on a Windows computer. We aim to show that if desired, smart home automation systems can still be built using simple viable alternatives to wireless communication.**

Keywords—Energy consumption; home automation; serial communication; microcontrollers

I. Introduction

Home automation started around 100 years ago when introducing electric power to domestic houses lead to the introduction of the first automated home appliances, such as the kettle in 1889 or the washing machine in 1904.

That automation process continues to this day as we need and also want ever more complex automated systems in our homes, making our lives easier, safer and more comfortable. Such systems are good for us and also for the Environment as they can significantly reduce our energy consumption by applying intelligent control to lighting, heating or power outlets in our homes.

It has been observed that in the United States, approximately 40% of the energy consumed was used in residential buildings [1] while in Europe the value was lower (but still significant), at 27% for the year 2017 according to Eurostat [2].

Multiple devices for controlling only one or two variables in the house and some complete home automation systems have already been released to the market. Installing them allows us to control the access, heating, lighting or the air conditioning (among others) but one important health related concern arises. In order to provide long distance control, each of these devices connects to the internet and to other devices inside the home mostly via Wi-Fi and Bluetooth.

We see light bulbs, thermostats, alarm systems, surveillance cameras, IoT hubs and multiple smart home appliances all interconnected and communicating over wireless networks, generating electro-magnetic (EM) radiation.

According to the Statista.com web-site [3], in 2010 the number of network connected devices / inhabitant, on a planetary scale was 1.84. By the year 2015 that number rose to 3.47 devices / inhabitant and the estimations showed that by the end of 2020, each person alive on the planet will own an average of 6.58 network connected devices.

The concern of over exposure to EM radiation is real, especially in the case of apartment buildings where in each flat that accommodates 2-3 people we can expect to find around 10 such devices and at any point in the building we can be surrounded by countless sources of EM radiation. This topic is rarely discussed and we tend to neglect the possible long term implication on our health.

Considering the above, we decided to look at different wired communication methods and developing one to be used within such automated systems, with the aim of reducing the EM radiation generated by the traditional wireless communication.

As this is a long term research project, in this paper we will focus on finding a simple and efficient wired and low radiation communication method between the microcontroller in charge of the automated system and a monitor and control app running on a personal computer.

The aim of future research will be to develop the full scale smart home automation system that will make life easy and comfortable for the residents, reduce the energy consumed in the building thus reducing the energy footprint and of course, limit as much as possible the amount of new EM radiation generated.

II. Literature Review

According to ABI Research [4] and to a study on Statista.com [5], in the United States the number of home automation systems went from 1.5 million in 2012 to 45 million in 2019 and the market value of these systems is expected to reach 12.81 billion USD by the end of 2020, a strong appreciation from 5.77 billion USD in 2013.

Such strong public interest towards these systems attracted multiple companies to finance their development and multiple research papers were published. Even so, the market is still fragmented and international standards in this domain are still being formulated. We studied several of these papers and will shortly discuss the findings and how our system will differ and where it will take a similar approach.

A general presentation of smart home automation devices and an outline of the advantages one has while living in a smart home is found in [6]. This paper proposes the use of X10 (wired) and ZigBee, Z-Wave and Insteon (wireless) technologies to achieve data exchange between the components of the system within the residence and the use of Ethernet for long distance access and control of the overall system. While the wireless communication inside the residence is something that we try to avoid in our system, the wired Ethernet connection is a good solution.

Another approach to smart home automation is presented in [7] and published in 2017. It is suggested to use an Android mobile phone that communicates via Bluetooth to an Arduino board in charge of switching the lights, air conditioning, smoke detectors and others. The main thing missing is the system's ability to act independently of the user and to make its own decisions. Further still, the use of Bluetooth technology and the need to keep the mobile phone close to the user even while indoor is something that we try to avoid with the system we develop. On the other hand, the use of an Arduino board is a good decision that makes for a cost efficient and scalable system.

As seen before, Bluetooth is a very popular technology in home automation. It is also used in [8] where a Bluetooth based client server network with its own custom built communication protocol is presented. The HAP- Home Automation Protocol is at the core of this system where a PC acts as server and the other sensors and electric devices are connected as slaves. Even though Bluetooth emits less radiation and communicates over shorter distances than Wi-Fi, it is still generating new EM radiation, something that we try to avoid. In addition to that, the system proposed in this paper lacks the ability to interact with the user over long distances, something that in our vision is compulsory.

Closer to our vision is the system presented in [9], where an Arduino Uno board is used to control the automated system and all the sensors and actuators are wired to the Arduino thus avoiding wireless communication. Even so, the system does have a mobile phone component, an app that connects via Bluetooth to the Arduino, something we have seen in most of the papers reviewed. Long distance communication with the system is in this case also not implemented.

All IoT devices do offer long distance monitoring and control. Philips Hue intelligent light bulbs or Fibaro smart power outlets, along with smart TVs, ACs and video monitoring devices can be connected to a hub such as Amazon's Alexa, Apple's Siri or the Google Nest Hub. Some people may be concerned however regarding how safe these devices are to cyber-attacks, as they are always connected to the internet. In [10] the issues associated with the security, privacy, safety and ethics of IoT devices are widely discussed and commented.

After reviewing the sources above, it was decided that an efficient smart home automation system can be built around an Arduino Uno board. This will make the system highly scalable and modular while keeping cost very low. All the sensors and actuators will be connected to the board by wires, thus limiting the amount of new EM radiation generated.

Most of the actions of the Arduino will be performed autonomously and instead of Bluetooth, for monitoring and configuration purposes we will connect to the board via the USB cable from a personal computer , which is of course the main topic of this paper and will further be discussed.

III. System Design and the Communication Method

Building a smart home automation system around an Arduino makes the task of adding, removing and communicating with sensors and actuators straight forward through the I/O pins. The challenge however, is establishing an efficient and reliable way to communicate with a control and configuration app running on a personal computer.

Arduino boards already connect via the USB port to the Arduino IDE (running on the PC). This is how new software is uploaded to the board. It is through the same USB cable and through the same ports that we will establish a serial communication between the Arduino board and a C# app, as outlined next in Fig. 1.

A. *Arduino Uno*

We will assume that the Arduino boards, and mainly the Uno board is already familiar to the reader and will focus only on the communication aspects related to the Arduino Uno. The board itself has a built in type B USB port and connects through cable to a type A USB port on the personal computer. If needed, communication to other devices is possible using built-in pins.

As seen in Fig. 2, the Arduino Uno can communicate to other peripherals using UART, I2C and SPI dedicated pins as it will be discussed next.

Unlike peripheral devices on traditional computers, on microcontroller boards peripherals are not necessarily independent devices, they can also be parts of the board itself that are dedicated to a specific tasks. These specific tasks are unrelated to the central processing unit itself and run independent of it. (E.g. a real time clock module may be integrated and run independently of the CPU but physically be part of the same board). Other peripherals may be connected as separate boards through the I/O pins. As with traditional computer peripherals, the ones on microcontroller boards have the same purpose: to make specialized tasks easier.

B. *I2C – Inter Integrated Circuit*

I2C is an asynchronous, multi-master, multi-slave serial communication protocol especially designed for microcontrollers in 1982 by Philips Semiconductor.

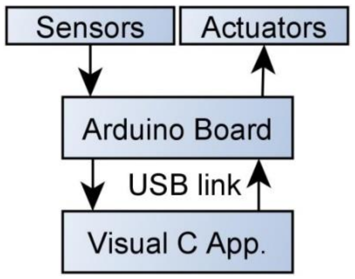


Fig. 1. Overview of the basic Data Flow between Components.

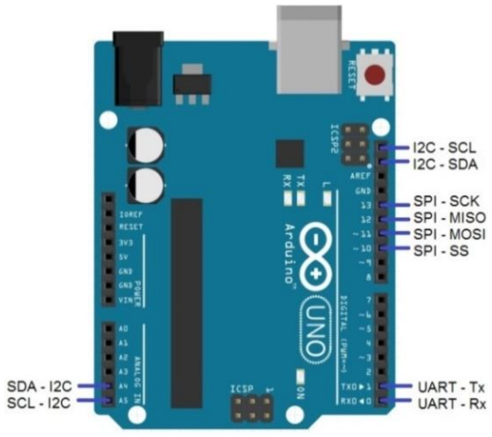


Fig. 2. The Communication Pins on Arduino Uno R3.

It is used for attaching lower speed peripherals to microcontrollers. It is very popular with modules and sensors in short-distance and often intra-board communication. Theoretically, using I2C communication it is possible to connect up to 128 devices to an Arduino board, more than enough for most projects, including ours.

When connecting a large number of modules to a microcontroller board, it is useful to consider the main controller board as the “master” and the other devices (sensor / actuator modules) as “slaves”. Because all these devices (masters and slaves) are connected to the same wires, maintaining clear communication among them is essential and such a task is possible by implementing an address system on the shared bus.

It is easy to understand why, in this configuration, all communication starts with specifying the unique address of the recipient device before sending data. The simplicity of the wiring used to implement this system comes with a slower communication speed trade-off, in comparison to SPI.

C. SPI – Serial Peripheral Interface

SPI was developed by Motorola in the mid ‘80s and similar to I2C, it is also a serial communication protocol. However, the similarities end there as SPI is a synchronous communication interface, developed for high-speed data exchange between one master device and up to four slaves.

As SPI is much faster than I2C, it is used in applications such as data logging on memory cards and displaying data on dedicated liquid crystal displays. The trade-off for this higher speed is the need to use dedicated wires for the four slave devices.

D. UART – Universal Asynchronous Reception and Transmission

The first thing to know about UART is that it is not a communication protocol. UART refers instead to the physical specialized integrated circuits found on the board and that allow for serial data to be send and received. This is the way our Arduino board communicates with the Windows app running on a PC so it will be presented in greater detail.

In UART communication, two devices exchange messages directly between each other while being physically connected by 2 wires. The first device converts the data to be sent from a parallel to a serial format and then it sends it towards the UART receiver. The receiver converts the received data from serial back to a parallel format so that it can be used by the device requiring the information. The two wires linking the UART circuits are connected as follows: the Tx pin of the transmitter connects to the Rx pin of the receiver, as seen in Fig. 3.

As UART asynchronously transmits data, there is obviously no clock to synchronize the transmission and reception of data packages between the two circuits. To overcome this drawback, the transmitter adds start (low state) and stop (high state) bits to each data package it transmits. These bits mark the beginning and the end of the data packages so that the UART receiver knows when to begin and finish reading data.

When a receiver detects a start bit, it begins reading the transferred data at a specific frequency known as “baud rate”, or transfer rate. This baud rate is a measure of the speed at which data is being transmitted between devices, measured in bits per second. Both UART devices, transmitter and receiver must use the same baud rate and the same structure of the data package. If the two devices use baud rates more than 10% apart, the communication between them is no longer possible.

As seen in [11], the baud rate between UART devices is generally set at 9600 bps, but it is possible to increase it up to 115000 bps.

Regarding the detection of errors during data transfer, this task is performed by including a parity bit in the data package. Factors such as the presence of strong electro-magnetic radiation, the use of different baud rates in the transmitter and receiver or simply the use of extremely long connection wires may affect the integrity of the data packages, leading to the occurrence of errors in the data. In such cases, the presence of the parity bit allows the UART receiver to verify if errors have occurred during data transfer and request a resend.

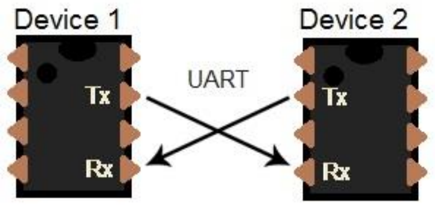


Fig. 3. Wired Connections between Two UART Devices.

The way this is done is quite simple. When a data package is received, the UART device counts how many bits in “high state” or “logic 1” it has received and checks if this number is odd or even. Then the parity bit is checked. If the total number of high state bits in the data package is an even number and the parity bit is zero (also indicating an even number), the UART receiver determines that no errors have occurred during the transfer of that particular data package. If this is not the case and a disparity between the number of high state bits and parity bit is identified, the receiver determines that an error has occurred during transfer and the data in that particular package is corrupt.

In a similar way, if there is an odd number of high state bits in the data package and the parity bit is 1 (also indicating an odd number), the receiver will again determine that no errors have occurred during transfer. As conclusion: errors have occurred when the parity bit does not match the odd or even nature of the number of counted high state bits in the data package.

To ensure a correct transmission of data, UART implements another check in addition to the parity bit. If the communication line is held in “logic 0” or “low state” for longer that the time needed to send a character, this fact will be registered as a break condition by the devices.

UART is a master-slave communication system that does not allow for multiple master devices or multiple slave devices to coexist in the same network. This constitutes a drawback in the case of complex systems but in the case of our project, it will do just fine. It should be noted that there is no perfect communication system or protocol, and UART is simple and efficient enough to still be in use and still very popular.

The fact that the structure of the data packages can change according to the needs of the application and the fact that only two wires are needed to link the two devices further constitutes an advantage and a reason to use this means of communication in our project and in many others as well.

E. The Communication Protocol

Just as the UART transmitter and receiver both use the same baud rate and the same data package structure in order to successfully exchange information, both software applications (the one uploaded in the Arduino board and the one running on the Windows PC) need to follow the same rules while exchanging data, otherwise the communication between them will not be possible. This set of rules implemented by both apps form a communication protocol.

In order to test and validate the chosen communication solution and the compatibility between components from the perspective of bidirectional data exchange, the following communication protocol was implemented in both the Arduino IDE sketch run by the Uno board and in the Visual C# app running in the Windows PC.

For now, this communication protocol is just a proof of concept and defines only several commands, but the structure of the data packages allow for close to 377000 commands to be defined and implemented in future versions if ever needed.

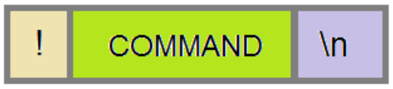


Fig. 4. The Structure of a Data Package.

As seen in Fig. 4, each transmission must begin with a specific character that signals the beginning of the data package. For this purpose we will use the exclamation mark “!” meaning “listen Arduino” followed by the command.

The command consists of 5 characters, letters or numbers combined as desired. These commands are issued by the Visual C# app as Master and executed by the Arduino board as Slave. At the end of the command there is a mandatory “\n” marker that signals the end of the data package and the Uno board executes whatever it was requested of it.

Several commands were defined and implemented in the two software applications communicating in our system. These commands allow the app. to connect and disconnect from the Arduino, to switch on and off its on board LED, to request the temperature and humidity values from a DHT11 module connected to the Arduino and display those on our PC and to request the board to write these values on a memory card. The results of their implementation will be presented in the Results and discussions section of this paper.

F. The Arduino Sketch

Arduino IDE (Integrated Development Environment) is an open-source software used to write, compile, debug and upload programs to Arduino boards. The code is written in C/C++ and several specific methods and functions are added.

The Arduino program, called “sketch”, can safely be uploaded onto the board if no errors occur during compilation and if the dynamic variables do not exceed the limited memory available.

The built-in flash memory of the board is non-volatile, so the sketch is not lost when the board is disconnected from the power supply. This way, when the board is repowered, it will automatically reload the sketch and run.

In order to test the communication link and protocol between the Arduino board and the Windows PC, a sketch was written for the Uno board and a Visual C# app for the Windows PC.

An overview of the Arduino sketch is presented in Fig. 5. It begins with a initialization part where libraries are included and variables are declared. For accessing the temperature sensor, the “dht.h” library must be included and to write data on the memory card the “SPI.h” and “SD.h” libraries are used. For the purpose of accurate time keeping the real time clock module on the Data logger shield is accessed and for this task, the “wire.h” and “RTClib.h” libraries are needed.

In the Setup function the serial communication is initiated and the baud rate is set, in our case at 9600 bps. The RTC is also initiated and set if needed (using the time and date the uploaded sketch was compiled at). For writing on the SD memory card, it is checked if the card is available or not and finally, the built-in LED is set as output so it can be turned on and off later on in the sketch if desired.

The main part of the sketch is the “void loop()” function. This part runs, as named, in a loop as long as the board is powered. At this point in the sketch the serial data is read. If the Start command is received, this will validate an “if” condition inside the loop and from that point onwards, other commands will be answered, until the “!Stoop\n” command is received.

Most of the sketch in this loop is composed of “if” conditions checking whether certain commands are received. For every command received one of these “if” conditions is validated and the appropriate instructions are executed (e.g. reading the temperature, switching the LED on/off, writing the SD card).

We did not include a “Serial.end()” function in this sketch, that means the Rx and Tx digital pins are always reserved for the UART communication and that our Arduino board is constantly ready to receive commands through the serial link.

G. *The Visual C# app*

Microsoft Visual Studio is an Integrated Development Environment – IDE used to develop apps capable of running on any platform.

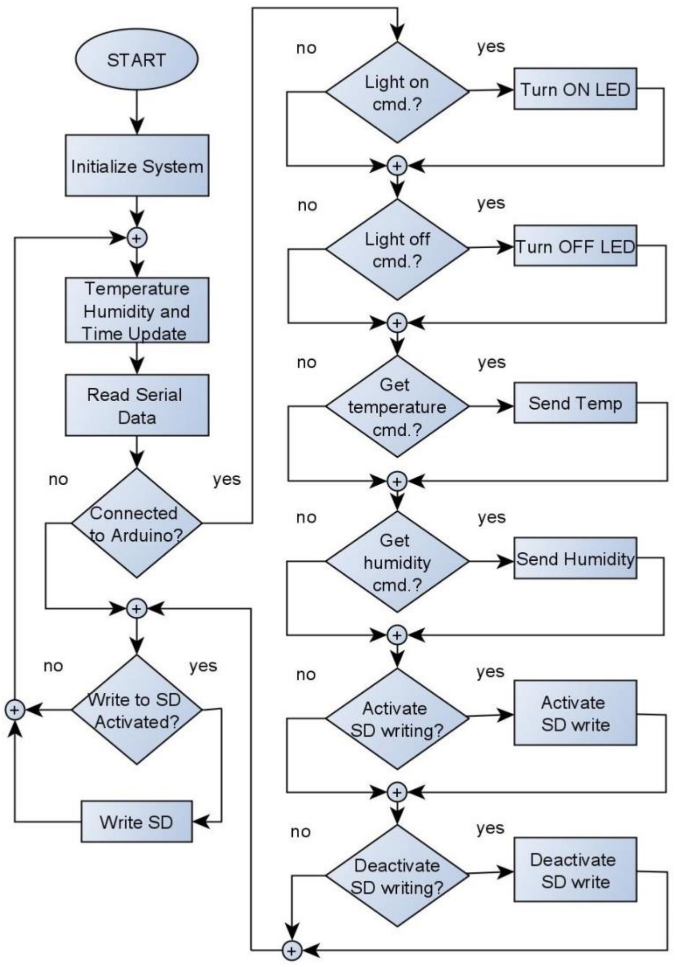


Fig. 5. Overview of the Arduino Sketch Logic.

We refer to Visual C# when Visual Studio is used to develop a C# app. It makes it easy to create modular applications where code can be reused, it has multiple libraries that make it easy to implement a multitude of functions and creating a windows form app is fast and straight forward. The decision to develop a Windows app was taken as Windows OS represents 77% of the global market in the area of desktop and laptop computers at the time this article was written, so it is relevant to the vast majority of users around the World.

Similar to the Arduino code, in the C# app we also set up the serial communication by selecting the communication port and baud rate and sending the “!Start\n” command to the Arduino board. While the two entities are connected, it is possible to send commands and receive data to and from the microcontroller. The specific examples will be discussed next, in the Results and discussion part of this paper. As a general rule, all commands are issued as text sent through the serial link and structured according to the communication protocol described earlier. All received data is simply read from the serial monitor and displayed in the text boxes of the Windows app. All data received from the Arduino is time stamped so that it is easier to keep track of it.

IV. Results and Discussions

While researching a simple and efficient way to exchange data between a microcontroller and a Windows running program, two apps were developed, a communication protocol set up and messages were exchanged successfully between them.

As seen in Fig. 6, the Arduino board has a data logger shield with a SD memory card attached and a temperature sensor also connected.

In this simple configuration there is no pin conflict detected, however, if more sensors or modules are connected, it has to be kept in mind that digital pins 0 and 1 on the Arduino board are used for the UART communication. In case these pins also need to be connected to other devices, the UART communication is harder to achieve, but not impossible and the Arduino board has to be programmed in such a way, as to take turns between using these pins for UART and sensor communication, each time opening and closing a serial communication link and then interacting with the sensor. The power to the sensor must also be turned on and off accordingly to avoid simulant data transmission towards the Rx pin. This complex setup however is not recommended.

The Arduino sketch was written in Arduino IDE and uploaded onto the board. As seen in Fig. 7, while compiling the current version it was noted that it used 17088 out of 32256 bytes of on-board storage memory (52%) and the variables occupy 1370 out of 2048 bytes of dynamic memory available (66%). There are sufficient memory resources still available to further develop the sketch and add to its functionality. For the Windows app, such details are not relevant as memory resources on desktop and laptop computers are plentiful.

The interface of the Windows app is presented in Fig. 8. When the “START” button is pressed, the “!START\n” command is sent to the Arduino board and from that moment on the board will accept other commands as well. Pressing the

"STOP" button will have the opposite effect, sending a "!STOOP\n" command and closing the communication link.

The "Read T" button sends a "!GETMP\n" command to the board and it will cause the Arduino to answer with a serial message containing the time and the temperature recorded by the DHT11 sensor module. This data is displayed in the window. This will be an "one off" event and to obtain another temperature reading the button has to be pressed again.

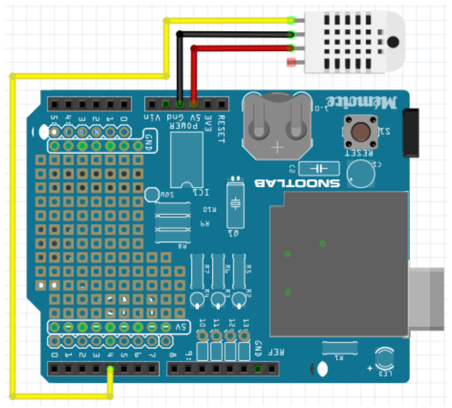
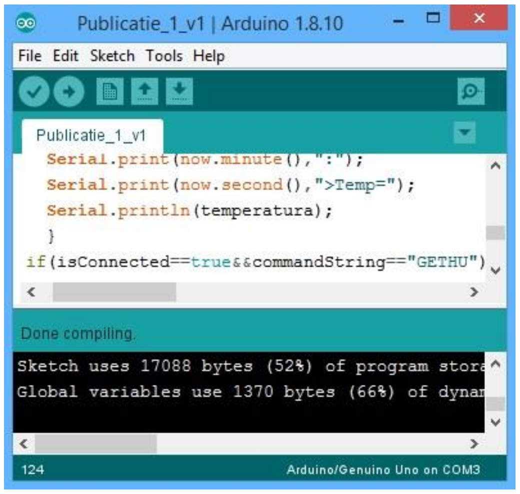


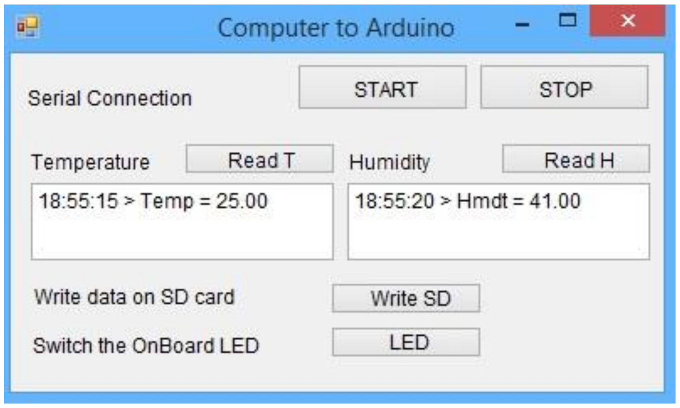
Fig. 6. Arduino with Data Logger Shield and Temp Sensor.



```
   Serial.print(now.minute(),":");
   Serial.print(now.second(),">Temp=");
   Serial.println(temperatura);
   }
 if(isConected==true&&commandString=="GETHU")
```

```
Done compiling.
Sketch uses 17088 bytes (52%) of program stora
Global variables use 1370 bytes (66%) of dynan
124                     Arduino/Genuino Uno on COM3
```

Fig. 7. The Arduino IDE Window and Sketch Code.



18:55:15 > Temp = 25.00

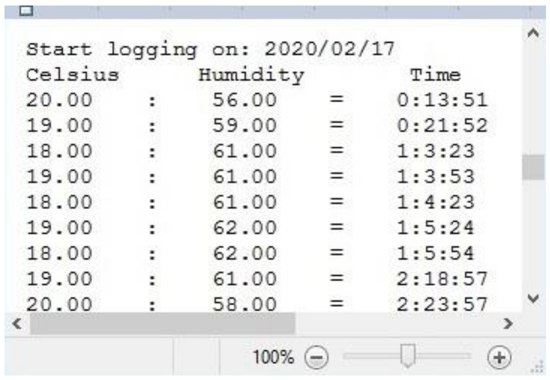
18:55:20 > Hmdt = 41.00

Fig. 8. The Graphic user Interface of the Windows App.

The system behaves in a similar way if the "Read H" button is pressed, with the exception that the "!GETHU\n" command is sent and the humidity value is received.

If the button "Write SD" is pressed, the command "!WRTON\n" is sent to the Arduino and data is being written onto the SD card inserted into the data logger shield. A sample of this data is seen in Fig. 9. If the same button is pressed again the "!WRTOF\n" command is sent to prompt the Arduino to stop writing data on the SD card. The app decides what command to send by checking the state of a Boolean variable that is changed every time the button is pressed, alternating between True/False, meaning write or stop writing.

Turning the on-board LED on and off is done in a similar way, the only difference is the command being sent that can either be "!LEDON\n" or "!LEDOF\n".



```
Start logging on: 2020/02/17
Celsius          Humidity          Time
20.00     :      56.00      =     0:13:51
19.00     :      59.00      =     0:21:52
18.00     :      61.00      =     1:3:23
19.00     :      61.00      =     1:3:53
18.00     :      61.00      =     1:4:23
19.00     :      62.00      =     1:5:24
18.00     :      62.00      =     1:5:54
19.00     :      61.00      =     2:18:57
20.00     :      58.00      =     2:23:57
```

Fig. 9. A Sample of Data from DHT11 Recorded Overnight.

If the sensor received data (such as temperature or humidity) was displayed on a dedicated LCD shield attached to the Arduino Uno, we would need 9 pins (out of 20) dedicated to this task. Other pins would also be required if we were to attach more buttons to activate or deactivate different functions of the board. All this would mean that the number of sensors and actuators that could still be attached to our board would be limited, something that is not desirable.

In contrast to the scenario outlined above, connecting the Arduino to a Windows PC and controlling it through an app, makes it an all-around better solution. The graphical user interface is easy to use, we can integrate countless buttons and commands without using any additional I/O pins and so more sensors and actuators can be attached to the Arduino board.

V. Conclusions and Further Work

The successful exchange of data between the microcontroller board and the Windows app is extremely important. It proves that it is possible to build a cheap and easy to program automation systems (Arduino based) and to connect it to a user friendly application for control and monitoring purposes.

Such a communication method does not generate any new EM radiation in the environment (as opposed to Wi-Fi or Bluetooth) and still offers full configuration and control capabilities over the microcontroller board.

This paper is an essential part in developing a complete Arduino based Smart Home Automation system that will make the home more comfortable to live in while having limited new EM radiation generated in the environment and very importantly, reducing the energy footprint of that home.

Our work continues with perfecting the architecture of this smart home automated system, selecting the modules to be included, the electrical connections between them and finalizing the control software, all of which will be made public in future papers over the coming year.

References

[1] Online resource: United States Energy Information Administration https://www.eia.gov/energyexplained/index.php?page=electricity_in_the_unit ed_states

[2] Online resource: https://ec.europa.eu/eurostat/statisticsexplained/index. PhpEnergy_consumption_in_households#Energy_consumption_in_ house holds_by_type_of_end-use.

[3] Online resource: https://www.statista.com/statistics/678739/forecast-on-connected-devices-per-person/

[4] Online resource: "1.5 Million Home Automation Systems Installed in the US This Year". *www.abiresearch.com*. Retrieved 2018-11-22.

[5] Online resource: https://www.statista.com/outlook/279/109/smart-home/ united-states#market-users

[6] R. John Robles and Tai-hoon Kim, "Applications, Systems and Methods in Smart Home Technology: A Review," International Journal of Advanced Science and Technology. 15: 37-48-2010.

[7] Ms. Poonam V. Gaikwad, Prof. Mr. Yoginath R. Kalshetty, "Bluetooth Based Smart Automation System Using Android", International Journal of New Innovations in Engineering and Technology, Volume 7 Issue 3– April 2017.

[8] N.Sriskanthan and Tan, Karande. "Bluetooth Based Home Automation System". Journal of Microprocessors and Microsystems, Vol. 26, pp.281-289, 2002.

[9] Theint Win Lai[#1], Zaw Lin Oo[*2], Maung Maung Than[*3],"Bluetooth Based Home Automation System Using Android and Arduino "[#]Faculty of Computer System and Technology, University of Computer Studies (Sittway),No.(123),Natmauk Road,Bahan Township, Yangon, Myanmar

[10] Hany F. Atlam and Gary Wills,"IoT Security, Privacy, Safety and Ethics", published on: March 2019,DOI: 10.1007/978-3-030-18732-3_8 In book: Digital Twin Technologies and Smart Cities Publisher: Springer Nature Switzerland AG 2020.

[11] Online resource: http://www.circuitbasics.com/basics-uart-communicati.