Cairo University
Faculty of Engineering

Department of Electronics and
Communications Engineering
ELC2 3030 *Spring2022*

UART & USB Report

Group 25

B.N.	Sec.	الاسم
23	1	احمد محمود حسيني
32	1	اسلام ناصر محمود
4	2	حسن احمد حسن السمان

Abstract

In this report we will explain a project we written that is a MATLAB code to emulate the protocol and the performance of a Universal Asynchronous Receiver Transmitter (UART) which is a serial to parallel or parallel to serial interface. Then we write another code to the Universal Serial Bus (USB) which is a serial bus, and it is commonly used and known. We finally plot samples of their output when they take some input and at some configuration of them.

Universal Asynchronous Receiver Transmitter (UART) Code

We start to import our configurations of the UART protocol such as (bit duration, parity …) from a text in JSON format named with “uartconf.json” file and use it mostly like using of a structure. We create a new variable to this configuration without using the variables of the json files to avoid any overlapping in the code and make it easy to use them. We take also an example of input text file named with “inputdata.txt” as double array of the American Standard Code for Information Interchange (ASCII) in one column. We convert then this double array to it is binary values and save it in a new array with dimensions of the data as in the configuration file and the input length.

After we read our configuration and the input and make them ready to use then we calculate the length of our channel that is depends on the number of stop bits and if there are a parity or not then we make an array to the channel and array for the holding register as in the UART protocol we take the data byte in one array and then shift it out one at a time. After initializing the lengths, we create a big nesting loop the big loop is from the first row in the data array to the end and take the raw and put it in the holding register and put the starting bit in the channel. The small loop then will shift the bits of the holding register in the channel bit by bit starting form Least Significant Bit (LSB) and it is also calculating the parity of this byte during the shifting then return to the big loop and make a condition if the parity is odd we invert it as our calculation for even parity and add it after the Most Significant Bit (MSB) in the byte or the word then another if condition to the number of bit in the stop condition if one or two bits and shift the channel and add them. Before opening the second iteration of the big loop the channel which now contain the start, data byte , parity if there and stop

bit/s will save in dynamic array with fixed number of rows equal to the complete channel and dynamic number of columns increasing by one after each iteration in the big loop until end of the data in the input file.

The last things we calculate in the big "for" loop the total time which is a counter increment after each loop with the bit duration multiplies by the number of columns in this row. In the same manner we calculate the total number of useful bits. We put all these flipped channels in a one array with one row to be the sequence of data transmitted with time to help us at end to plot the time diagram (TD) to sample of two bytes.

At the end of loop we have the total time and total useful bit from then we calculate the efficiency which equal the total number of useful bits in the channel over the total number of bits in the channel. The overhead then will be equal to one minus the efficiency.

$$Efficiency = \frac{\text{Useful Bits}}{\text{Total Bits}} \quad , \quad Overhead = 1 - Efficiency$$

Output sample of UART:

After calculating we plot a sample of two bytes in the input file with time that is in (figure 1).

When the first 2 binary bytes in input data if data bit in the configuration 8 is:

'01010100'

'01101000'

And bit duration = 0.0001 second. Parity is = 'none'.

So, after adding the start, … etc

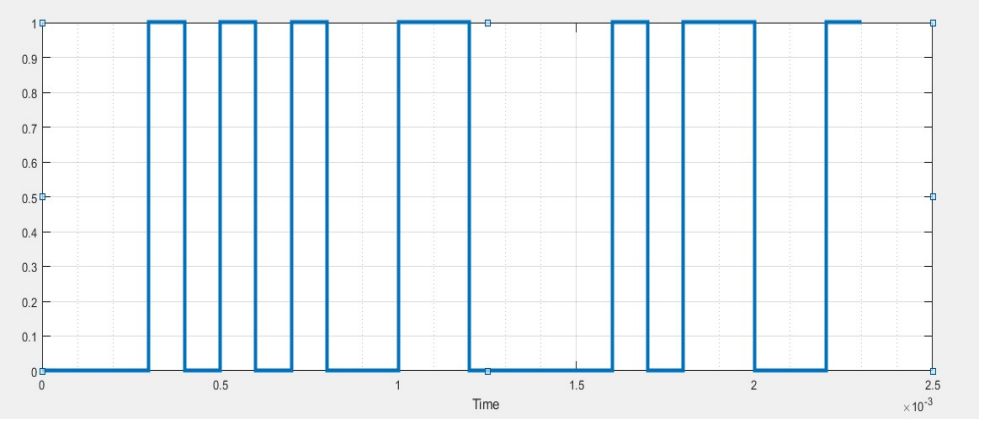

Figure 1 :UART Time diagram for sample of 2 bytes.

The first clock pulse will be the start bit = 0, then the LSB = 0, then 0, … then the MSB = 0 then, the stop bit= 1 . then the second byte with clock in the same manner.
The total time in this case was = 1.279999999999970 , Overhead = 20 % , Efficiency = 80%
Which mean for all 10 bits the useful bits or data bits only 8 bits.

First, we make this code separately and track it to be executable at all critical cases then we start at a separate code for USB.

Universal Serial Bus (USB) Code:

We start the USB code almost like the UART code we read the configuration file with JSON extension and save it in a new variable then read the same input file and make it ready to use by dividing it into rows equal to the packet length and columns equal to the number of packets in the input file. Then we create a big matrix to put the synchronization pattern (after repeating it with the number of complete packets) at first then contracting it with the packet identifies (PID) which is a 4 unique bits and the other 4 is the complements of them then we contact them with the matrix of data (after we separate the last packet if is there which length is smaller than the complete packet as it can not be in the same matrix).

Now, we have all packets in each columns with the synchronization pattern, PID so we create a nesting “for” loop to add the stuffing bits and make the inverting packet (NRZI) the big “for” loop will repeat with the number of packets then a small loop for each packet to check if there are a series sequence of six one and put a zero after then another small for loop to make the packet after adding the stuffing bits NRZI by checking if the next bit is one don not change the last bit and if the next bit is zero invert the last bit. After these small two loop we make a dynamic array to save the length of the packet as if there are stuffing bits adding the length will differ. At the end we save the packet in one array with one column as we know it is length and add the end of packet which will be three zeros. At the next iteration of the big for loop we put the next packet in the same columns after the end of the last packet and so on for all packets.

After we transmit all packets, we do the same manner to the last not complete packet. We create array of one column and put the SYNC pattern then the PID after calculating it then the last not complete packet bits then we make the same nesting loops to add stuffing bits and make NRZI and add the end of packets EOP at the end.

The all packets now in arrays of one columns after adding SYN pattern, PID, … so the total time of the all packets will be the bit duration multiplying to the size of these arrays (complete packet array, not complete packet array). The efficiency and overhead calculation will be the same.

$$\text{Efficiency} = \frac{\text{Useful Bits}}{\text{Total Bits}} \quad , \quad Overhead = 1 - \text{Efficiency}$$

Output sample of USB:

We plot the time diagram to sample of the first 2 packets that is in figure 2,3.

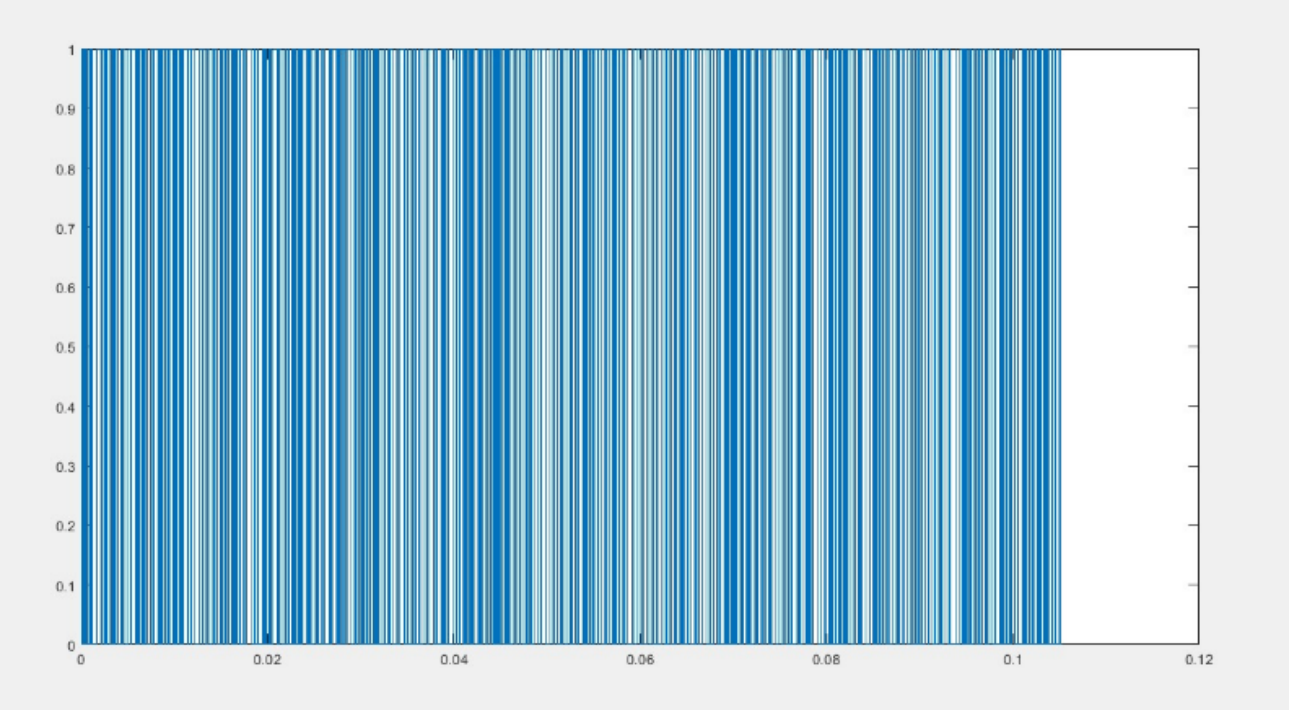

Figure 2Time Diagram to Sample of 2nd packet

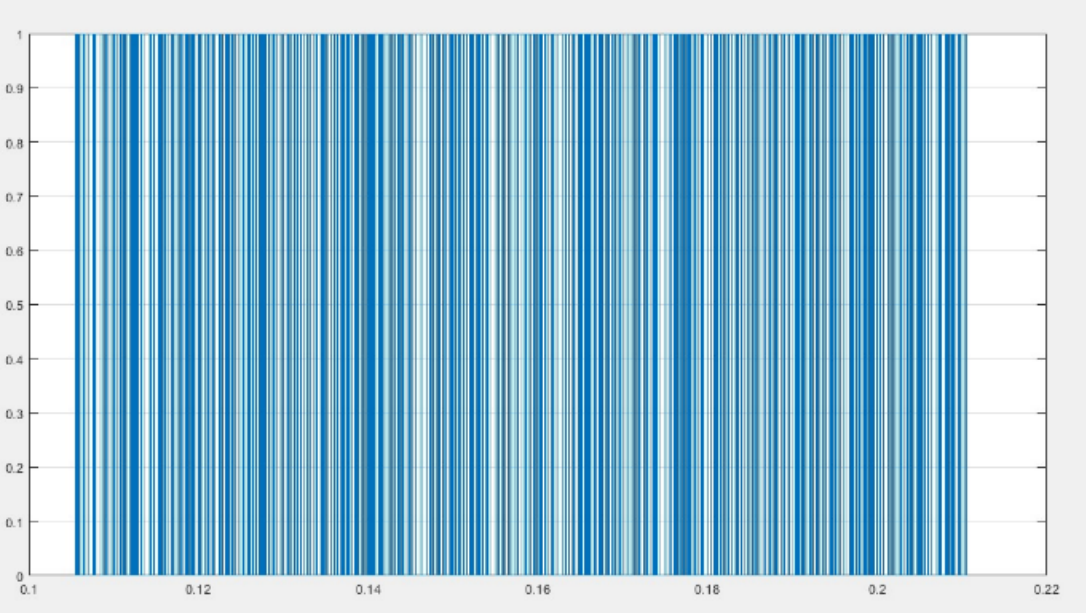

Figure 3Time Diagram to Sample of 1st packet

Note that in time diagram as we plot a complete packet it is not easy to track it but we made that in the code by checking the array for first 2 bytes and last 2 bytes in the packet and it was easy to track it for the last not complete packet.

The total time in the case of the same input file was = 1.054000 sec ,Overhead = 02.846 % , Efficiency = 97.15%. Which mean for all 100 bits the useful bits or data bits about 97 bits.

Conclusion

We notice that for the same file input the USB is more efficient than UART by +17.15% and which mean it is total time of USB is less than for UART and its Overhead is less than UART Overhead. USB is more efficient UART as it transmits packet by packet not bit by bit as in UART interface

Final Code:

After we wrote 2 separate codes, we made a on code that read the 2 configuration files and put them in one configuration file with extension JSON and make switch to find if this conf file has a protocol name and run the code of this protocol as we make them as functions which returns the final values of parameters.

Then we create a condition if the duration time of each protocol are equal so we call a function we created to plot sample of each protocol and plot the total time and overheads of each protocol versus the increasing of the input file size and we make the increasing by doubling the input file from one to ten times it and that is the plots in figures 4,5.

When we plot the overhead in the two cases we found that for the same configuration input file in UART. The overhead in USB is a constant but it because the input file does not have any stuffing bit.

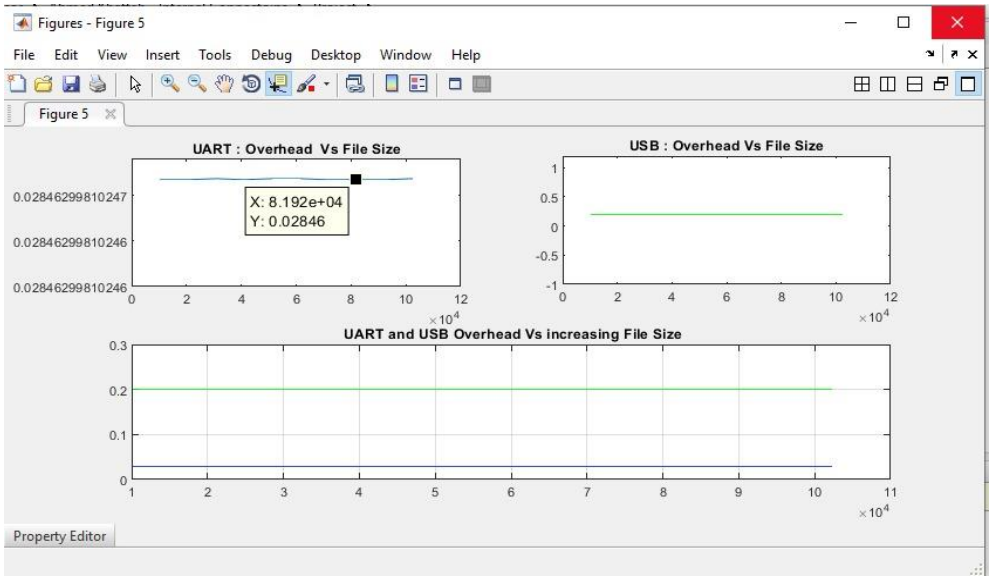

Figure 4

We note there the increasing in input file will cause increasing in total time of the UART more than the USB.

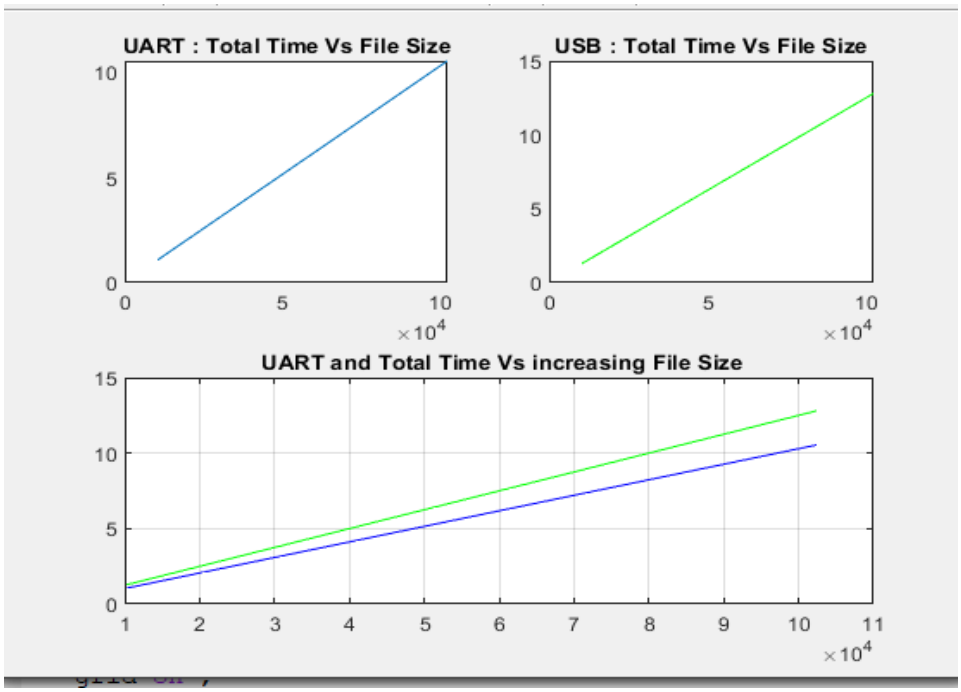

Figure 5

The input file does not have a six sequence of ones so the overhead in USB does not affect so to see this change we will put random stuffing bits by put "~" many times in random places in the input file and plot their Overheads and Total time in figure 7.

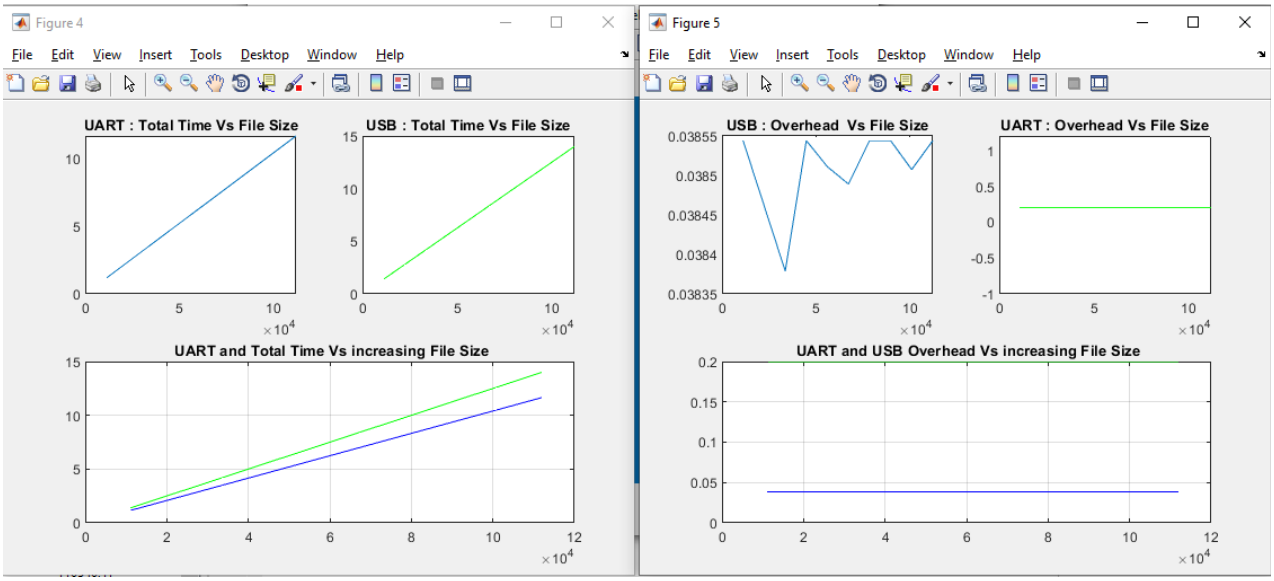

Figure 7

we notice that the increasing in data size will cause a non-linearity in the overhead as it will change from size to another.