

**INTRODUCTION**

The report is about an open ended design exercise to implement the Transmit of the **RS232** it was the first milestone reached in this journey. It was a degree for mechanical device typewriters and modems for digital data exchange introduced in 1962 by the Radio Sector of EIA. It made the data exchange more reliable over analogue channel. The standard outlined voltage levels that created it proof against immune to noise disturbances and reduced the error in data exchange. Today, it is still available in the form of COM ports of most PCs

And when we connect something to a computer it likely uses USB (Universal Serial Bus) or various card slots such as SD cards, tiny micro SD cards and a number of other formats. Unlike more modern developments in serial data transmission to and from computers, like USB and FireWire, RS232 is comparatively uncomplicated, and thus easy to implement during a hardware laboratory or simulator environment. The electric signalling levels and methods are handled by a driver circuit external to the UART. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port.

**HARDWARE DESIGN**

A little introduction to what this hardware design is going to cover is that, the system is essentially a parallel to serial converter, and can work in the following sequence.

* Capture data byte to be transmitted onto an 8-bit register
* Output one start bit, starts from (0000).
* Output data bits 0 to 8 (in that order, i.e. LSB first)
* Output one (or more) stop bits when the binary number reaches (1001) – this is the RS232 quiescent or 'resting' state.
* Done.

The whole circuit was meant to be designed on the Logic boards.

**3.II- Parallel to serial conversion**

Digital parallel and serial converters adapt data transmitted in a parallel port to a serial communication format, or adapt serial port data to parallel communication format.

Data converters are needed because not all devices in a system work entirely as parallel-only or serial-only components. For instance, the communication within a computer uses parallel buses that transmit parallel information from element/component to component/element. There are many possible ways of parallel-serial conversion.

* Serial-in, Parallel-out shift register
* Serial-in, Serial-out shift register
* Parallel-in, Parallel-out shift register
* Parallel-in, Serial-out shift register

**3.II.a- Types**

**3.III- Formal Design Methodologies**

**3.III.a Designing circuit**

* I loaded the data into a parallel-in, parallel-out shift register and designed the 4 bit counter which starts from decimal number 0 (0000) and stops at the decimal number 9 (1001).





Next step is to connect to the 74153N device that is activated when its (0), for 1G this refers to EN on the logic board and 2G in it. So the first 4 inputs from (0 to 3) depend on 1G so when one 1G reads (0) the device gets activated and the 4 inputs give an output. Same thing goes with the 2G as well. The two outputs 1Y and 2Y are connected to and OR gate so either way when its 1 in any of the inputs the output is always 1. Here 0 and 1 are connected to Vout1 & Vout2.

* Is to create a 2TO1 multiplexer using the logic gates. This multiplexer has 2 inputs (I0 and I1) so the inputs are (0) and (1) because it’s connected to the switch 8. So the enable G gets activated when it reads 0 from vout4 which is connected to an inverter and then it gives an output for the two inputs.
* The output from the 74153N and MUX\_2TO1, are connected to the logic gates as said in [5], the use of OR gate and a NOR is that the bits get inverted and stays at 1 and the output signal is shown on the screen in the logic analyzer.

**3.III.b Karnaugh map & Truth Tables**

The logic analyzer clearly shows that it stops at binary (1001) and remains there.

Based on these three different enabler 1G, 2G, G the truth table below shows it all from decimal number 0 to 9.Also shows when it’s getting activated and deactivated

**Multiplexer Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vout4 Vout3 Vout2 Vout1** | **G1** | **G2** | **G3** |
| **MUX1**  **0 0 0 0**  **0 0 0 1**  **0 0 1 0**  **0 0 1 1** | **0** | **1** | **1** |
| **MUX2**  **0 1 0 0**  **0 1 0 1**  **0 1 1 0**  **0 1 1 1** | **1** | **0** | **1** |
| **MUX3**  **1 0 0 0**  **1 0 0 1** | **1** | **1** | **0** |

**The device gets activated when it reads (0).** So that’s why when MUX3 is (0) for G3 the device is activated and at that time MUX1 and MUX2 are not active. Hence the output stops and stays on the decimal number 9.

**3.IV- Use of Higher Level Logic Constructs**

The reason I used **multiplexers** is because as said in [2] the multiplexing technique that is designed in a way to reduce the number of electrical connections or leads in the display matrix. Besides reducing the quantity of severally independent interconnections, multiplexing additionally simplifies the drive electronics, reduces the value and provides direct interface with the

microprocessors.

[**C HYPERLINK "https://en.wikipedia.org/wiki/Counter\_(digital)"ounter**](https://en.wikipedia.org/wiki/Counter_(digital)) is a sort of memory (and generally displays) a sole number of times a selected event or method has occurred, usually in relationship to a clock signal. Can be Asynchronous or Synchronous. They are also considered in junction with Finite State machines (FSM), more about FSM is said in [2].