

# Serial Data Transmission and Reception with UART

Noah Gardner, Peter Semrau, William Strand

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

e-mails: nbgardner@oakland.edu, pasemrau@oakland.edu, wwstrand@oakland.edu

**Abstract**—UART stands for universal asynchronous receiver/transmitter and is a circuit that sends parallel data through a serial line. A UART contains both a transmitter and a receiver. UARTs are useful because without such techniques, data would often be contained in their specific circuit without any useful methods for importing/exporting data to other circuits and devices. The goal of our project is to effectively show proof of concept so our design may allow others to implement our findings in their future projects.

## I. INTRODUCTION

This report will cover the methodology and all of the problems that had to be solved to finish the circuit, as well as our experimental findings and conclusions about what we have designed. The goal of our project is to be able to receive a string of data, and then echo the same data back to the computer to show transmission. We will first require the use of the terminal program PuTTY to interpret the key strokes from the computer, and then transmit the corresponding hexadecimal ASCII code back to the board through a serial line. The serial data will be loaded into a shift register until all the data has been loaded, then it will shift out the data in parallel, sending it to eight LEDs and a seven segment display. The LEDs will show the hexadecimal ASCII code of whichever keyboard character was pressed, and then the seven segment display will display the character if it is 0-9 or a-f. As for transmission of data, the UART transmitter circuit that was provided in class is implemented, but instead of the switches being used for the input, it will be the same parallel output from the shift register.

This project required both some knowledge covered in course material, but the majority of the challenge was the independent research. The transmission of data was largely covered in class, with the circuit and code provided for us. However, the reception of data is not covered in class and coursework, and required a lot of independent research.

## II. METHODOLOGY

### A. Adjusting for Baud Rate

The baud rate refers to the rate at which information is transferred in a communication channel. For our project, the internal clock of the Nexys 4 board is not in sync with the clock on the computer. The Nexys 4 clock runs at 100MHz and the serial port of the computer can transfer data at a maximum of 9600 bits per second. This means that the internal clock on the Nexys 4 board is much faster than the

rate at which the computer will send and receive data, and these two clocks must be in sync with each other in order to communicate. This problem is solved by implementing an additional counter that starts at the moment the data communication begins. The counter is to simply output a high whenever the count resets, and then the high is used as rate the board uses to send or receive the serial bits from the computer.

### B. Receiving Data

This part was the most challenging portion of our final project. The receiver essentially shifts in data bit by bit into a shift register, then reassembles the data. The data consists of one start bit, eight bits of data (the ASCII code), and one stop bit, which get transmitted from the computer terminal to the board. We went about managing this data by creating a finite state machine (FSM) with 12 states that shifts the data into the shift register. The first state waits for the start bit, and sends it to state two once it is received. State two then enables the two counters My\_genpulse and HalfCount. My\_genpulse keeps the board in sync with the transmission rate of the computer by changing the states whenever the counter finishes, while HalfCount samples the received bit halfway through each bits transmission and then shifts it into the register [1]. States 3-10 function the same way with each bit received. State 11 then waits until the transmission has ended by waiting until only 1's are detected because the default output of the computer is active high. State 12 enables the transmission section of the code while disabling HalfCount and My\_genpulse.

### C. Transmitting Data

The circuit for this part of the project is covered in class and in notes, and the same process is being implemented in our design. The circuit is primarily a shift register that loads data in parallel and then shifts it out one bit at a time at a specific rate. One major difference in the design however, is that the circuit uses a counter to transmit the correct number of bits of data, while the receiving circuit used the states in the FSM to receive the correct number of bits. The only change we made to the design of the transmitter provided in the notes is that we used the data from the receiver as the input for our transmitter. This will cause the transmitter to always echo back the same data that is received. Once the data has finished transmitting the interior circuit sends a done signal to the top level FSM and changes the top level state back to state 1.

### III. EXPERIMENTAL SETUP

In order to troubleshoot and verify that our project does indeed work, we programmed eight LEDs to display the output from the receiving shift register at all times. We used our knowledge of how the data was transmitted with the low start bit and eight data bits to interpret the mistakes that we were making in the code. It is now verified to be working as intended because the LEDs show the correct hexadecimal ASCII output whenever a key on the keyboard is pressed. A VHDL testbench was also used as a tool to help us visually verify our program. The behavioral simulation of the program would clearly show us what data was being loaded into the register at any time. The testbench was also useful for determining the timing on our data sampling counter, as you could visually see how the data sampling counter lined up with each bit of data.

### IV. RESULTS

This project most importantly verifies that parallel data can be effectively transmitted and received through a serial line. It also shows that counters can be used to synchronize clocks and read a fast moving string of data. This project confirms more basic topics that we have learned about in class, such as the creation and implementation of shift registers, counters, and finite state machines, as well as how to apply these components in VHDL.

A link to a video of our project functioning is provided here:

<https://www.youtube.com/watch?v=Ih5DEnTnat0&feature=youtu.be>

A link to the complete circuit diagram is provided here:

<http://i.imgur.com/pJj5pcA.png>

A link a behavioral simulation of our testbench is provided here:

<http://i.imgur.com/oFiO8wy.png>

Referring to the behavioral simulation, CState represents the state of the FSM we have created. When Zdone is high the state changes, and when Zhalfdone is high the data is sampled and shifted into the register. The signal leds shows what is currently loaded into the shift register at all times, and the bits can be seen shifting in one by one as the data gets sampled by zhalfdone.

### CONCLUSIONS

All of our goals as far as proof of concept have been achieved, and most importantly the challenge of receiving parallel data through a serial line has been accomplished. That being said, there are more creative ways to expand on our findings. For example, we considered saving the data received in past keyboard presses onto the other seven segment displays, giving the capability to form words. Unfortunately, we fell short on time and were unable to try this ourselves, but we still accomplished everything we initially set out to do. The ability to use a keyboard to send hexadecimal signals to the Nexys 4 board opens up a lot of possibilities, and the next step for our project would be to use it in a creative and interesting way.

### REFERENCES

- [1] Pong P. Chu, "FPGA Prototyping by VHDL Examples," Xilinx Spartan – 3 Version. John Wiley & Sons, New Jersey, pp. 163-181, 2008.