# We Process

Jonathon Helms & John Falvey
Electrical and Computer Engineering
School of Engineering and Computer Science
Oakland University, Rochester, MI
jkhelms@oakland.edu & jpfalvey@oakland.edu

***Abstract*—We Process is a project containing a combination of a microprocessor and a UART interface.**

## I. Introduction

We Processes is a combination of a microprocessor and a UART, outputting to two 7segment displays to show two hex values. These hex values are determined by the processors ALU, which obtains its values from the UART.

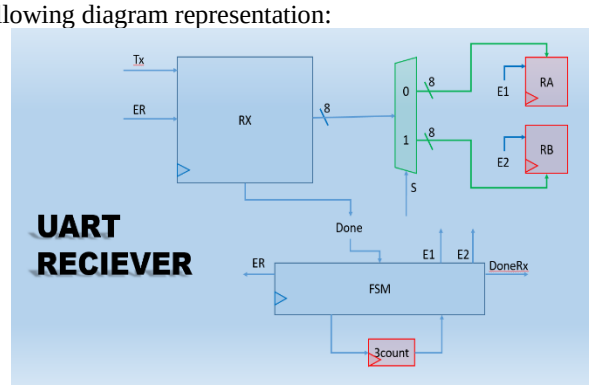
For the main dive, of We Process, practicality was the primary idea. Keeping the project within this realm of practicality meant looking for the most optimal use. The most optimal use, that was thought of, came from understanding how common serial communication is within industry, such as CAN systems. Serial communications interfaces, such as PuTTY, Visual Studio, or Java. As We Process was coming to fruition, it was observed to have a commonality between it and a calculator; giving the project not just a use as a microprocessor, but a use at a calculator which also contained logic functions.

## II. Methodology

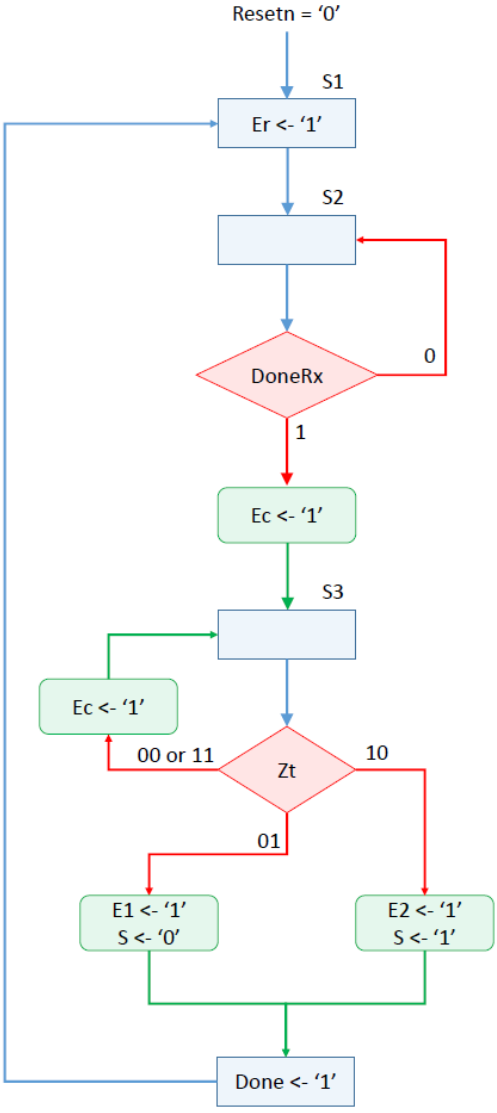
### *UART*

Possibly the most challenging part of the project; was the creation of an interaction between the computer and the NEXYS 4 DDR. Originally, the idea was to build our own UART using the knowledge that the serial communication information was coming out of the USB port with a start bit of 0 and an end bit of 1. However, matching the baud rates ended up being more challenging than previously expected.

The difficulty in matching the baud rates eventually drove the project to use/adapt example code to a form usable by the NEXYS 4. [1] [4] The code that was used suggest the following diagram representation:

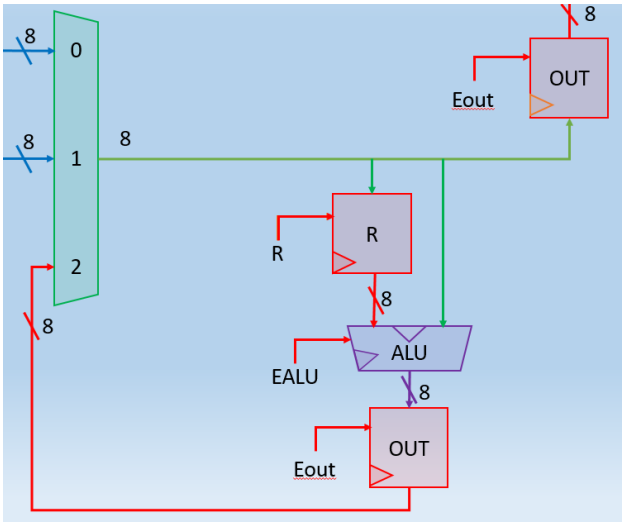



In the UART Receiver, there is are six modules that make up the system. The RX component outputs it data to the multiplexer and is then controlled by the FSM to send the data to the two registers. These the FSM has received the "done" signal from the RX receiver then the registers connected to the multiplexer shifts out the hex value to their designated locations. Following is the states that they FSM, for the UART, will go through during this operation:



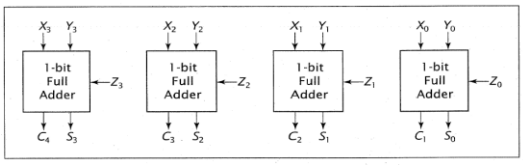
*Microprocessor*

The main body, in order to complete the desired operations and logic comparisons, required a multiplexer, register, FSM, and a ALU. The FSM would dictate how the data received from the UART was going to be handled. Once the UART was transmitting the data, the FSM of the microprocessor would know what the user wanted done with the given data values. Once the ALU was done with its operation, the outputted value is stored in an output register and transmitted to the 7segment displays. The diagram for the microprocessor, minus the FSM, is as follows:

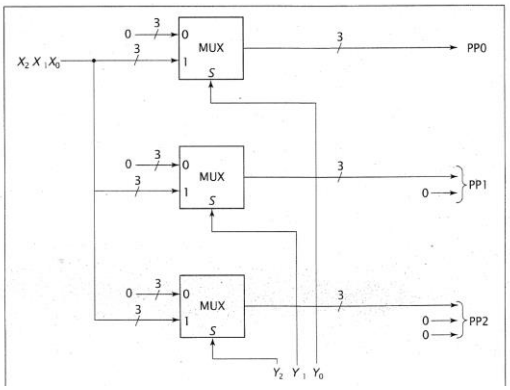



The ALU in the diagram above preforms a range of operations; from logic operators to multiplication. One of the major points with this ALU is the way multiplication is performed. [2] In order to create the circuit for the multiplier, instead of a ripple adder, which is taught in the class, this ALU used a Carry Save Adder along with a Wallace Tree. Their diagrams are as follows:

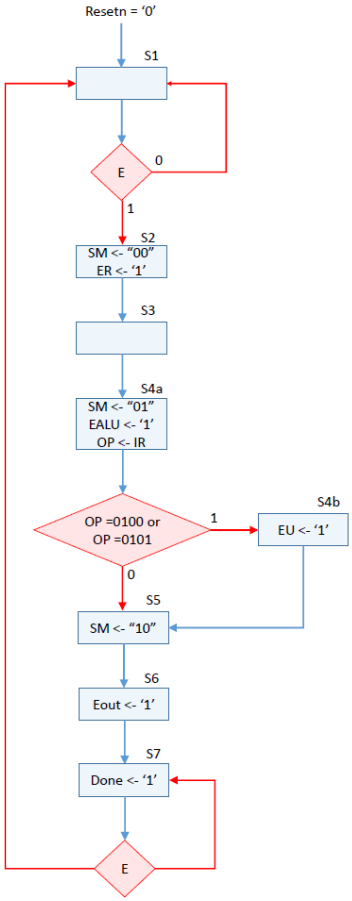
A carry-save adder




Generating partial products for multiplication using Wallace trees



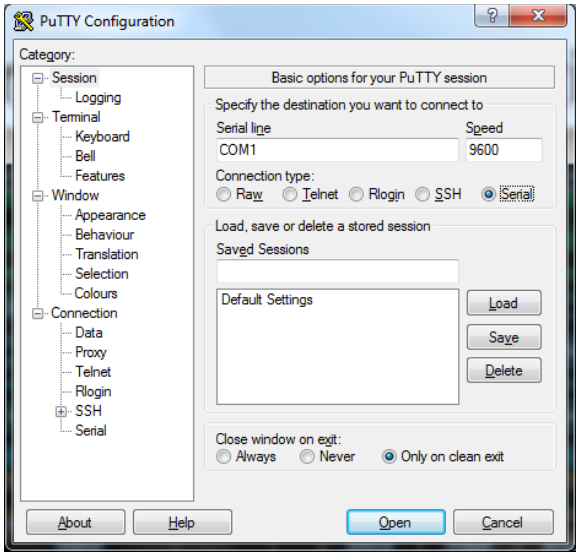

Due to the way this program was written, an additional signal had to be implemented in order to capture the resulting value. This value ended up being output about two clock cycles later than any of the other operational values. The controlling FSM for the processor was according to the following state diagram:




For the data to be transmitted to, the 7segment displays are always displaying the current value they have stored. When the program is first booted up; the values that are displayed on the board are zeros. Once the value being transmitted to the 7segment display is changed, the 8bits coming in are separated into two, four bit numbers, and their hex values are displayed on the displays.

## III. Experimental Setup

Along with a timing simulation, PuTTY was used to verify that the program would output the proper values (a timing simulation for the shift left register is provided on the last page). The PuTTY setup was designed to mimic the baud rate of a standard USB2.0 connection. The following PuTTY configuration was used for the project:




## IV. Results

Once the base of the project was finished, one of the many points that were noted was the change from using hex values as our output and changing it to dec. Another change that could have been bad, would have been adding more operations to the ALU, along with adding more registers and having multiple ALUs in order to perform multiple operations at once.

## Conclusion

The project showed that there are multiple, including easier, ways to perform the same operation. Using the Wallace Tree seemed to be much easier than the multiplier that was showed in the book.

Also, seeing the difficulties that are associated with matching baud rates between the board and the computer. This part of the project was extremely difficult. The baud rate was something that was really not able to be observed easy, therefore designing software that would work with it was challenging.

However, challenging the project was, using the tools that were learned during the class were helpful. Tools such as the state flow diagrams gave a great visual representation of how the program needed to flow. Combining the results of many timing diagrams, the state diagrams gave a better insight on when we needed to trigger different signals and when enables needed to be sent.

Given more time, we wanted to make the ALU and main body of the project more complex; however, once the project was operational with what we had built, it seemed too late to make any major changes to the design to fit what edits we wanted to make.

A take away from the project/class would have to be how modular in nature VHDL seemed to be. Being able to implement old code and diagraming out the program was useful, and made much of the project's design easier.

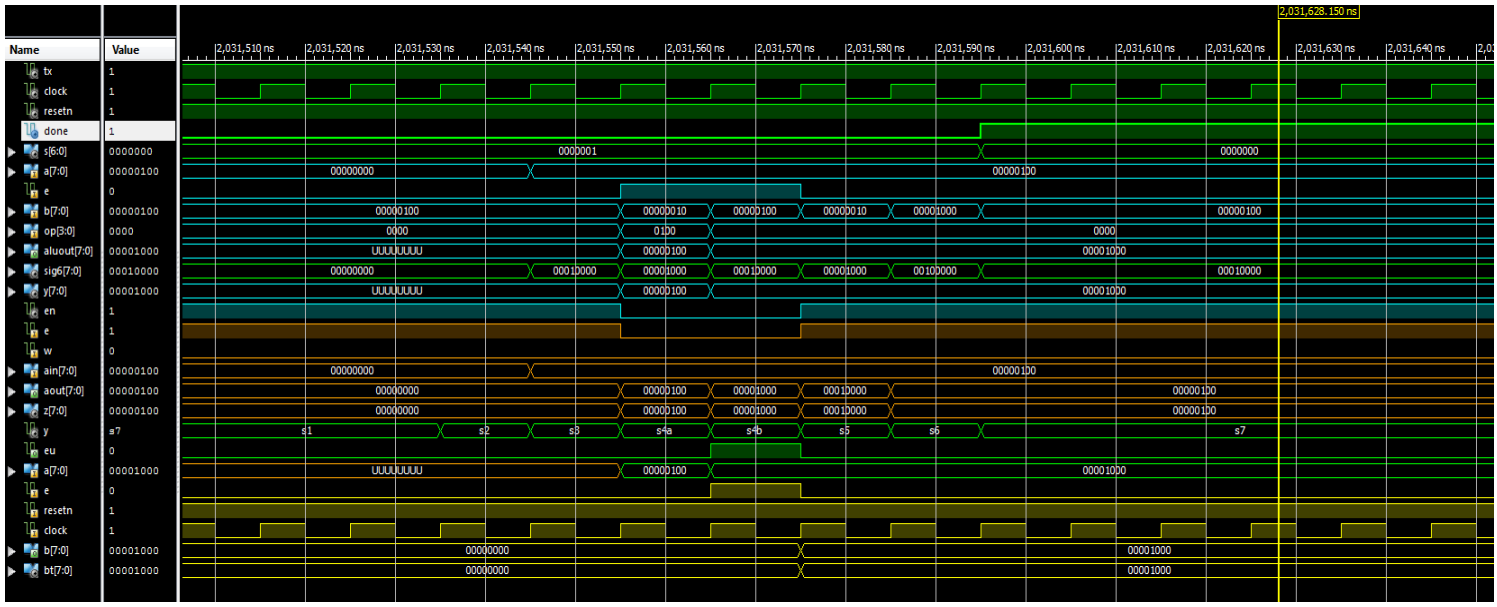
## References


[1] Chapman, Ken. "UART Transmitter and Receiver Macros." (n.d.): n. pag. Jan. 2003. Web. 6 Apr. 2016.

[2] Chu, Pong P. "FPGA PROTOTYPING BY VHDL EXAMPLES." (n.d.): n. pag. *Ebooksclub*. JOHN WILEY & SONS, INC., 2008. Web. 6 Apr. 2016.

[3] Cuzeau, Bertrand. "VHDL - Practical Example - Designing an UART." (n.d.): n. pag. *Javadoc*. ALSE, Sept. 2001. Web. 6 Apr. 2016.

[4] Toomey, Warren (DoctorWkt), and Hamster. "VHDL UART RX for Nexys4 DDR?" *Digilent Forum*. N.p., 9 July 2015. Web. 17 Apr. 2016. <https://forum.digilentinc.com/topic/766-vhdl-uart-rx-for-nexys4-ddr>.


2,031,628.150 ns
Name
Value
2,031,510 ns
2,031,520 ns
2,031,530 ns
2,031,540 ns
2,031,550 ns
2,031,560 ns
2,031,570 ns
2,031,580 ns
2,031,590 ns
2,031,600 ns
2,031,610 ns
2,031,620 ns
2,031,630 ns
2,031,640 ns
tx 1
clock 1
resetn 1
done 1
s[6:0] 0000000 0000001 0000000
a[7:0] 00000100 00000000 00000100
e 0
b[7:0] 00000100 00000100 00000010 00000100 00000010 00001000 00000100
op[3:0] 0000 0000 0100 0000
aluout[7:0] 00001000 UUUUUUUU 00000100 00001000
sig6[7:0] 00010000 00000000 00010000 00001000 00010000 00001000 00100000 00010000
y[7:0] 00001000 UUUUUUUU 00000100 00001000
en 1
e 1
w 0
ain[7:0] 00000100 00000000 00000100
aout[7:0] 00000100 00000000 00000100 00001000 00010000 00000100
z[7:0] 00000100 00000000 00000100 00001000 00010000 00000100
y s7 s1 s2 s3 s4a s4b s5 s6 s7
eu 0
a[7:0] 00001000 UUUUUUUU 00000100 00001000
e 0
resetn 1
clock 1
b[7:0] 00001000 00000000 00001000
bt[7:0] 00001000 00000000 00001000