CECS 460 System on Chip Design

Full UART Chip Specification

Jesus Franco

014046368

Fall 2020

12/16/2020

Revision History

Revision	Date	Changes
1.0	11/21/2020	Initial creation of document for a Transmit Engine to work with the TramelBlaze Processor.
1.1	11/28/2020	Added Receiver Engine portion to create a Full UART to work with the TramelBlaze Processor.
1.2	12/8/2020	TSI block was added so students could be exposed to what the industry does design wise. Courtesy of Professor Tramel.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. Introduction

This chip specification report is designed to give a detailed explanation on the UART project and its interface with the TramelBlaze processor. All blocks of logic and code for this project will be explained in detail. Simulation will also be provided for the main blocks to show verification in the design's functionality.

1.1 Purpose

The purpose for this report is to specify the requirements and details for the Transmit Engine and its interfacing with the TramelBlaze processor. This report contains details on the design implementation, design blocks for the design, and source code for all blocks to create the design. Each block will contain a description and a schematic to show the design. Some blocks will also be equipped with a table to map inputs and outputs.

2. Applicable Documents

2.1 TramelBlaze

The TramelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze. The ISA of the TramelBalze is the ISA of the PicoBlaze. The assembler for the TramelBlaze is written in Python and will generate the memory image files required for building a Xilinx code ROM along with other files useful for debugging.

2.2 TramelBlaze Architecture

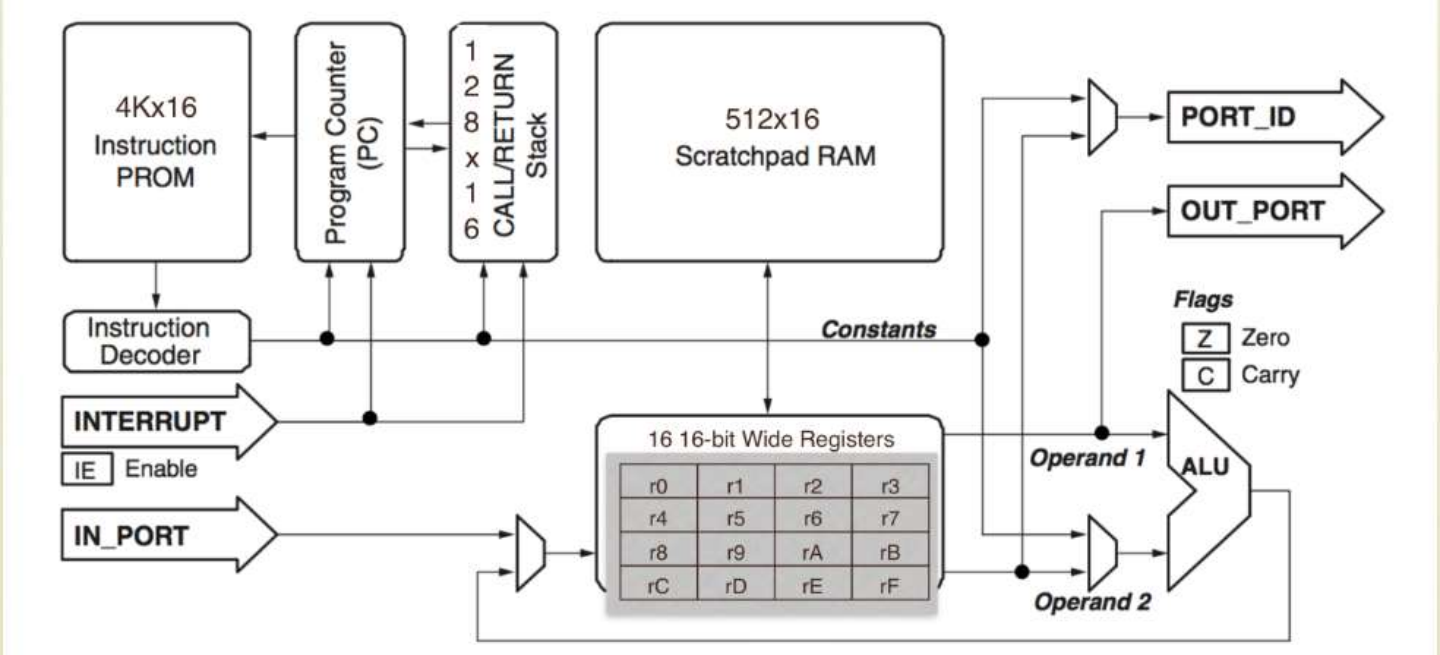

Figure 1: TramelBlaze Architecture

2.3 TramelBlaze Block Diagram

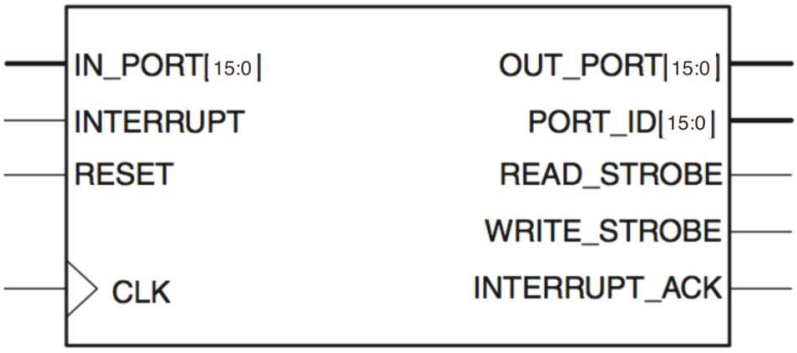

Figure 2: TramelBlaze Block Diagram

2.4 TramelBlaze Functional Blocks

Contains General-Purpose Registers – 16 16-bit wide registers from R0 to Rf.

4096 Instruction Program Store – the code ROM provides storage for a good number of instructions

Arithmetic Logic Unit(ALU) – the 16-bit wide ALU performs all microcontroller operations that include addition, subtraction, AND, OR, XOR, arithmetic comparisons, bitwise test operations, comprehensive shifts, rotate operations, program control operation such as jump or call, ZERO/CARRY flags, interrupt enable flags, 512-word Scratchpad RAM, INPUT/OUTPUT, a program counter, and a program flow control.

3. Requirements

3.1 Interfacing Requirements

BAUD[3:0]	RATE	BIT TIME	ENG NOT	N4 COUNT	N4 BITS
0000	300	0.003333333	3.3333 ms	333,333	19
0001	1200	0.000833333	833.33 us	83,333	19
0010	2400	0.000416667	416.66 us	41,667	19
0011	4800	0.000208333	208.33 us	20,833	19
0100	9600	0.000104167	104.16 us	10,417	19
0101	19200	5.20833E-05	52.083 us	5,208	19
0110	38400	2.60417E-05	26.041 us	2,604	19
0111	57600	1.73611E-05	17.361 us	1,736	19
1000	115200	8.68056E-06	8.6806 us	868	19
1001	230400	4.34028E-06	4.3404 us	434	19
1010	460800	2.17014E-06	2.1701 us	217	19
1011	921600	1.08507E-06	1.0851 us	109	19

Table 1: Baud Rate Select Values

The Full UART requires a baud rate from the given table above, courtesy of professor Tramel. The baud rate is controlled by 4 of the onboard switches from the Nexys A7 board. The Full UART requires switches[7:4] to be the selected inputs to control the baud rate. For this design and for verification purposes instructed by professor Tramel, all baud should have been tested and working functionally for the Full UART.

EIGHT	PEN	OHEL	DESCRIPTION
0	0	0	7 Bits, No Parity
0	0	1	7 Bits, No Parity
0	1	0	7 Bits, Even Parity
0	1	1	7 Bits, Odd Parity
1	0	0	8 Bits, No Parity
1	0	1	8 Bits, No Parity
1	1	0	8 Bits, Even Parity
1	1	1	8 Bits, Odd Parity

Table 2: Select and Descriptions for the Decode Module

The Transmit Engine requires a decode module to check for parity of the design and the total bits. For this design, 3 other inputs are mapped to control whether it is 7 or 8 bits, parity is enabled, and if the parity is even or odd. Input EIGHT is to determine whether it is 7 or 8 bits. If EIGHT is low, then the decode module will use 7 bits; however, if the EIGHT is high, then the decode module will use 8 bits. Input PEN refers to being the parity enabler. If PEN is low, then there is no parity; however, if PEN is high, then there is parity. Lastly, input OHEL determines whether the parity is even or odd. Assuming PEN is high and OHEL is low, then there is even parity; however, if PEN is high and OHEL is high, then there is odd parity. Otherwise if PEN is low and OHEL is either high or low, then there is no parity since PEN needs to be high to determine a parity.

3.2 Hardware Requirements

SW7	SW6	SW5	SW4	SW3	SW2	SW1
BAUD[3]	BAUD[2]	BAUD[1]	BAUD[0]	EIGHT	PEN	OHEL

Table 3: Switch Mapping for the Transmit Engine

This table shows the 7 inputs that are mapped to the onboard switches from the Nexys A7 board.

4. Top Level Design

4.1 Description

The top-level implementation of this design is meant to show the Receiver Engine and the Transmit Engine and its interface with the TramelBlaze processor as a Full Universal Asynchronous Receiver Transmitter (UART). Here it shows the top-level inputs such as the clock, reset, RX, and the switches. As well as the top-level outputs such as the LEDs and TX. Lastly, the Full UART will have all I/O ports be manually mapped through input and output buffers in the Technology Specific Interface (TSI) module.

4.2 Block Diagram

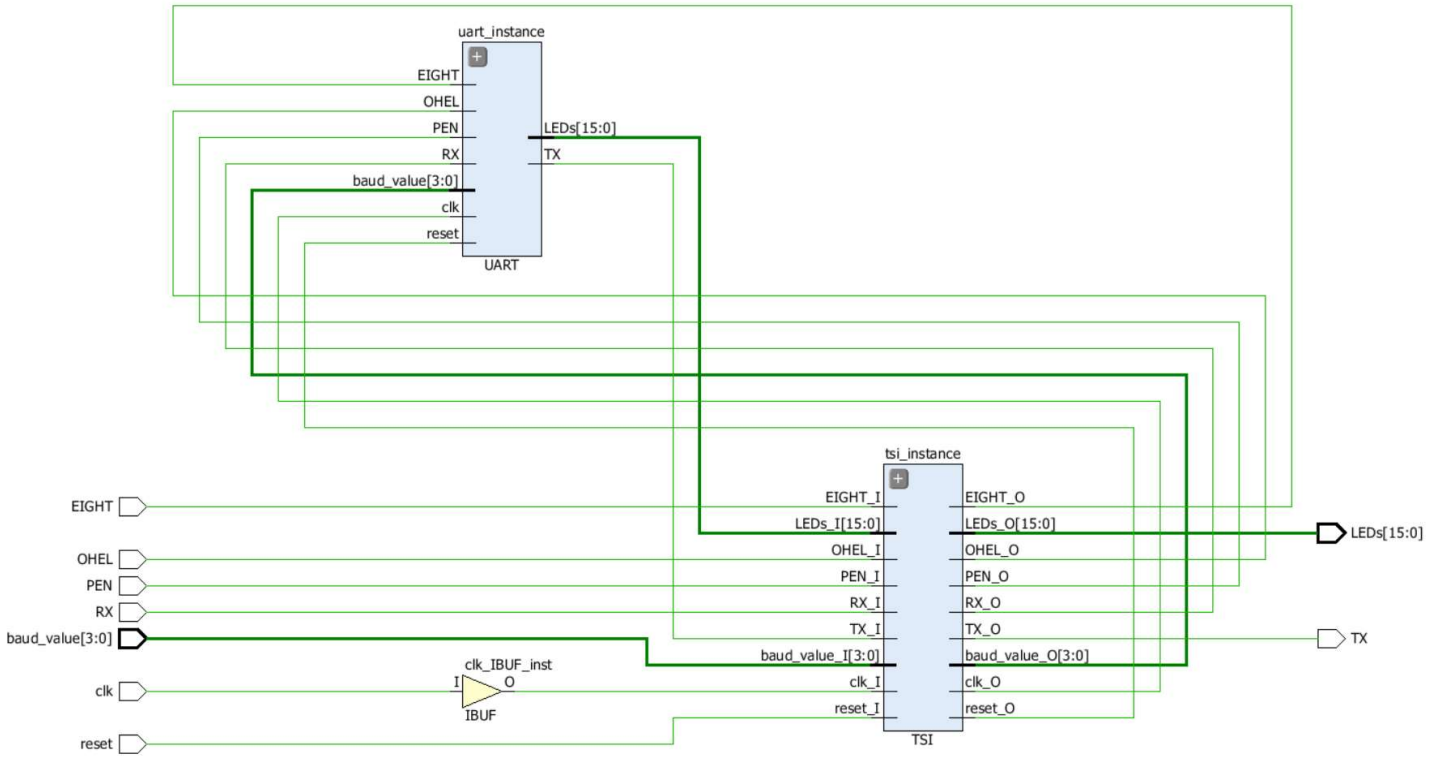

Figure 3: Full UART TSI Block Diagram

4.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 Board
reset	1	Input	Synchronous reset
RX	1	Input	1 bit received data from the receiver engine
baud value	4	Input	4 switch input for baud select
EIGHT	1	Input	1 switch input for the eight bit enable
PEN	1	Input	1 switch input for parity enable
OHEL	1	Input	1 switch input for even/odd parity enable
LED	16	Output	16 onboard Nexys A7 LEDs
TX	1	Output	1 bit transmitted data from the transmit engine

Table 4: Full UART TSI I/O

5. TSI Design

5.1 Description

The TSI is used to manually map I/O ports to buffers. Even though Vivado usually does this part for us, it is for us students' knowledge and for design practice since it is used in the industry.

5.2 Block Diagram

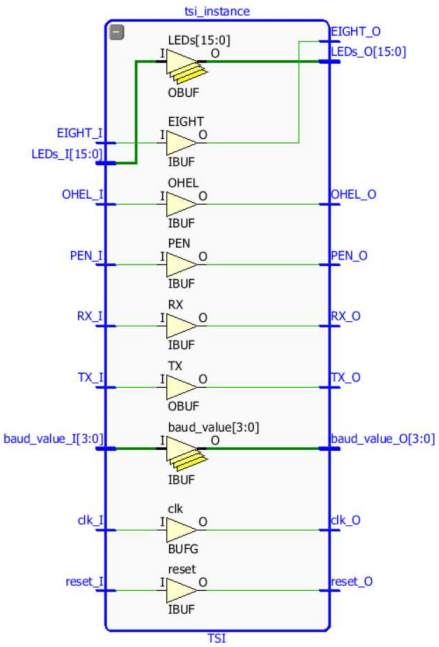

Figure 4: TSI Diagram

5.3 I/O Table

Signal	Bits	I/O	Description
clk_I	1	Input	100MHz clock from the Nexys A7 Board
reset_I	1	Input	Synchronous reset
RX_I	1	Input	1 bit received data from the receiver engine
baud_value_I	4	Input	4 switch input for baud select
EIGHT_I	1	Input	1 switch input for the eight bit enable
PEN_I	1	Input	1 switch input for parity enable
OHEL_I	1	Input	1 switch input for even/odd parity enable
LED_I	16	Output	16 onboard Nexys A7 LEDs
TX_I	1	Output	1 bit transmitted data from the transmit engine
clk_O	1	Input	100MHz clock from the Nexys A7 Board
reset_O	1	Input	Synchronous reset
RX_O	1	Input	1 bit received data from the receiver engine
baud_value_O	4	Input	4 switch input for baud select
EIGHT_O	1	Input	1 switch input for the eight bit enable
PEN_O	1	Input	1 switch input for parity enable
OHEL_O	1	Input	1 switch input for even/odd parity enable
LED_O	16	Output	16 onboard Nexys A7 LEDs
TX_O	1	Output	1 bit transmitted data from the transmit engine

Table 5: TSI I/O

6. Full UART Design

6.1 Description

The top-level implementation of this design is meant to show the Receiver Engine and the Transmit Engine and its interface with the TramelBlaze processor as a Full Universal Asynchronous Receiver Transmitter (UART). Here it shows the top-level inputs such as the clock, reset, RX, and the switches. As well as the top-level outputs such as the LEDs and TX.

NOTE: The source code could be found at the end of this Chip Spec in the appendix section under the name of UART.v

6.2 Block Diagram

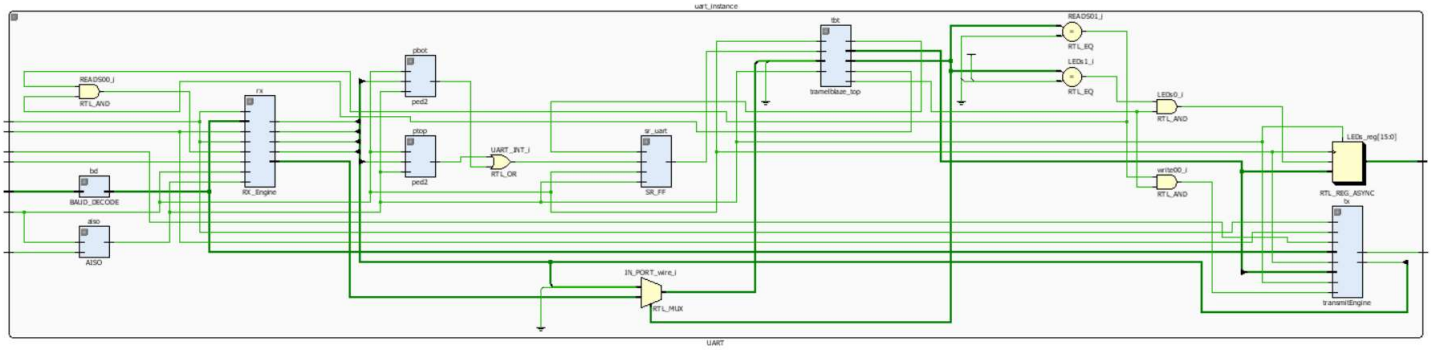

Figure 5: Full UART Block Diagram

6.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 Board
reset	1	Input	Asynchronous reset
RX	1	Input	1 bit received data from the receiver engine
baud_value	4	Input	4 switch input for baud select
EIGHT	1	Input	1 switch input for the eight bit enable
PEN	1	Input	1 switch input for parity enable
OHEL	1	Input	1 switch input for even/odd parity enable
LED	16	Output	16 onboard Nexys A7 LEDs
TX	1	Output	1 bit transmitted data from the transmit engine

Table 6: Full UART I/O

7. AISO Design

7.1 Description

The AISO takes in the clock and an asynchronous reset. The AISO will use 2 D flip flops as synchronizing flops where the first flop's input will be a 1 and then output a wire to go to the next flip flop as an input. The next flip flop will then use that input and make it its output. Lastly, the output from the 2nd flop will be negated for the synchronous reset and finally will output our asynchronous reset with synchronous de-assertion.

7.2 Block Diagram

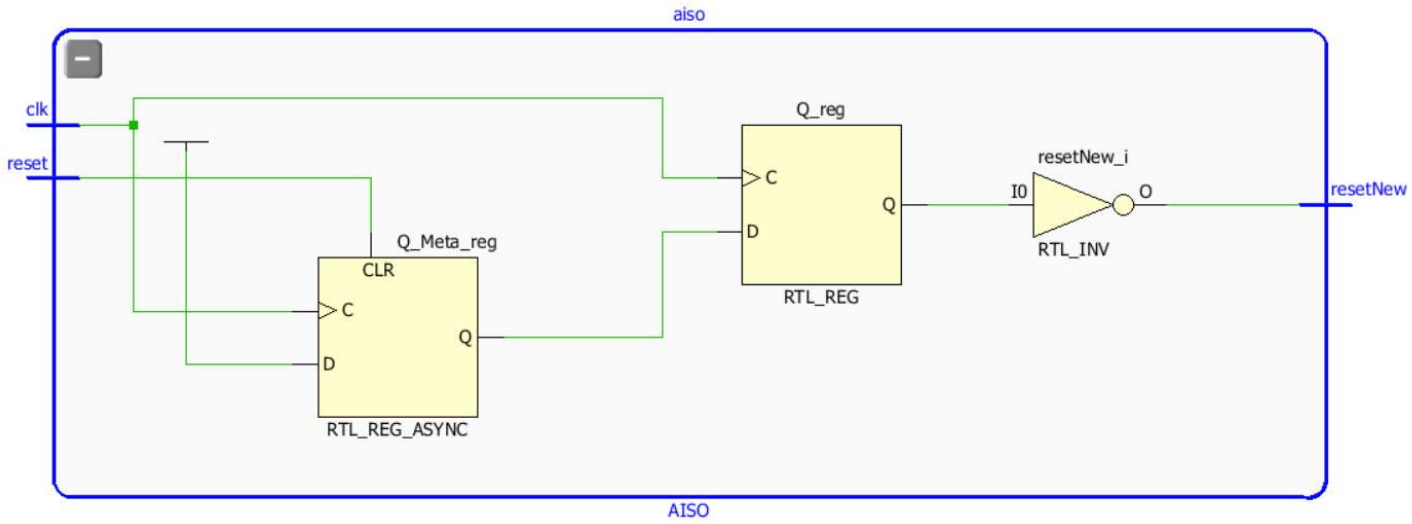

Figure 6: AISO Block Diagram

7.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Synchronous reset
resetNew	1	Output	Asynchronous reset with synchronous de-assertion

Table 7: AISO I/O

8. Positive Edge Detect Design

8.1 Description

The positive edge detect will output if it detects an active high signal. The positive edge uses sequential logic in terms of 2 D flip flops. The first flop will take in the output from the debounce as an input and create a wire to the 2nd flip flop for synchronization and an AND gate for comparing. The 2nd flip flop will then use that input and negate it to go into our 2nd input of our AND gate for the combinational logic. The result of the AND gate will be the positive edge detect output that will later go into the rest of the modules.

8.2 Block Diagram

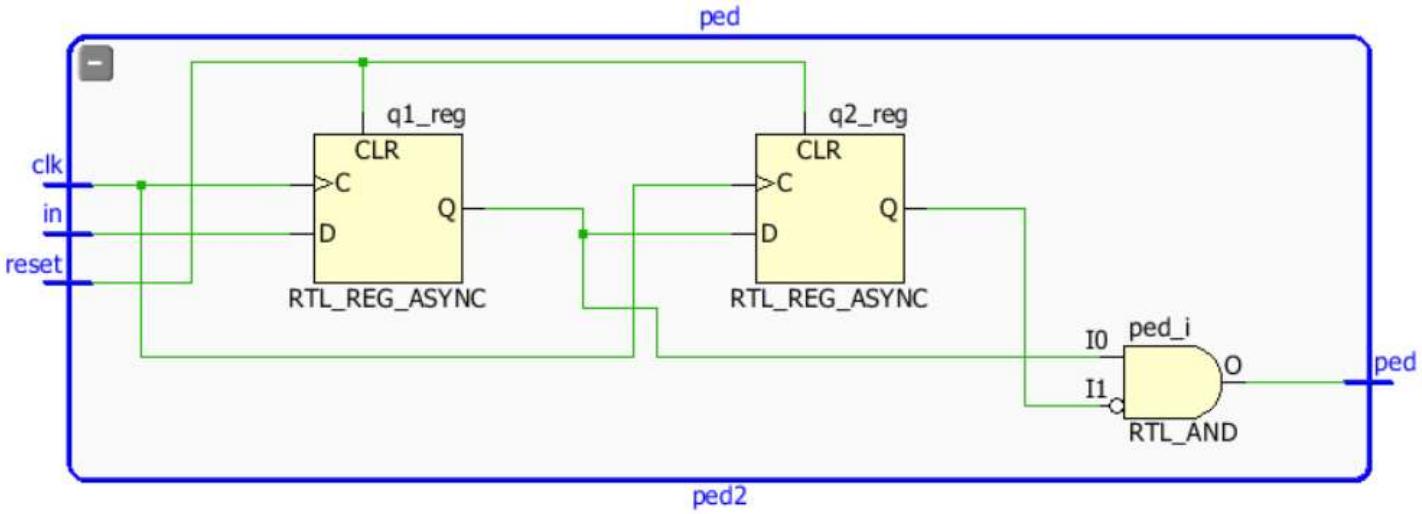

Figure 7: Positive Edge Detect Block Diagram

8.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-asssertion
in	1	Input	Incoming input
ped	1	Output	Postive edge detect output

Table 8: Positive Edge Detect I/O

9. SR(Set/Reset) Flip Flop Design

9.1 Description

The SR flip flop will take in the 100MHz clock and the synchronous reset with asynchronous de-assertion as their inputs for sequential logic. The S of the SR flip flop is meant to be SET and it refers to setting the output to '1'. The R of the SR flip flop is meant to be RESET and it refers to setting the output to '0'.

NOTE: This Project uses the SR Flip Flop module multiple times for the design.

9.2 Block Diagram

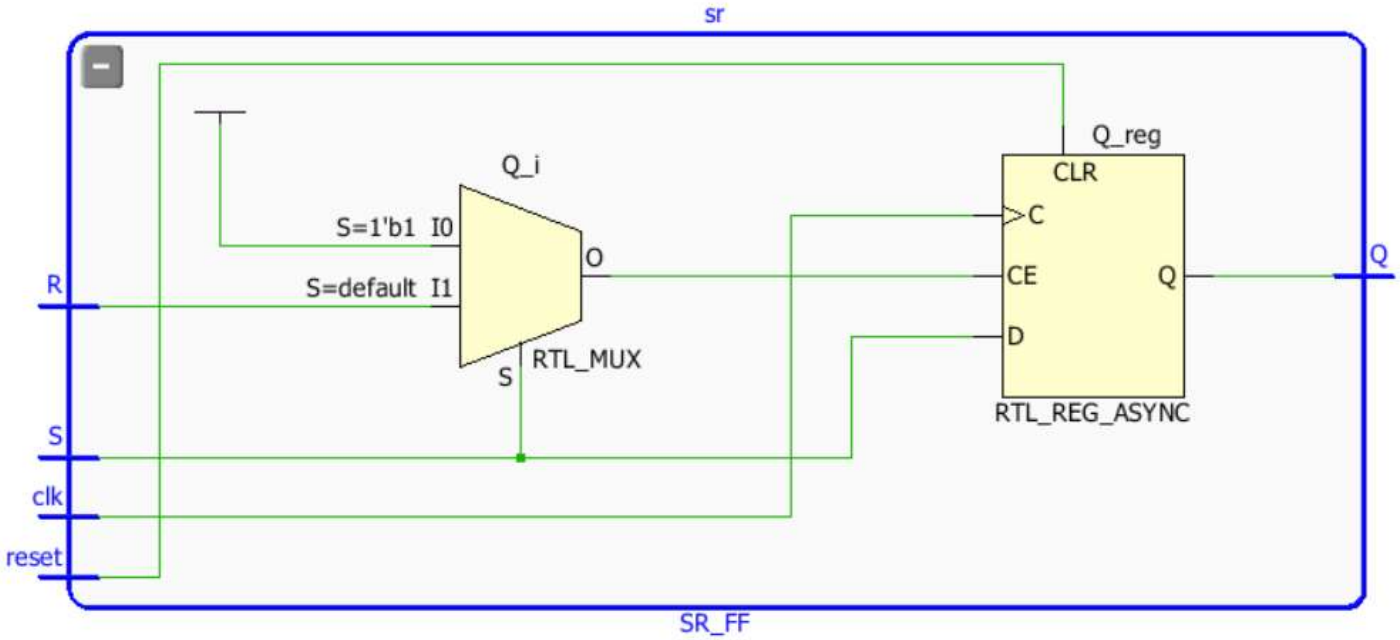

Figure 8: SR Flop Block Diagram

9.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion(will output a '0')
S	1	Input	SET input of the SR flop
R	1	Input	RESET input of the SR flop
Q	1	Output	Output of the SR flop

Table 9: SR Flop I/O

10. Baud Decode Design

10.1 Description

The Baud Decode module will determine the selected baud rate that will go directly to the Receiver Engine and the Transmit Engine. The baud rate will be controlled by the user manually on the switches.

10.2 Block Diagram

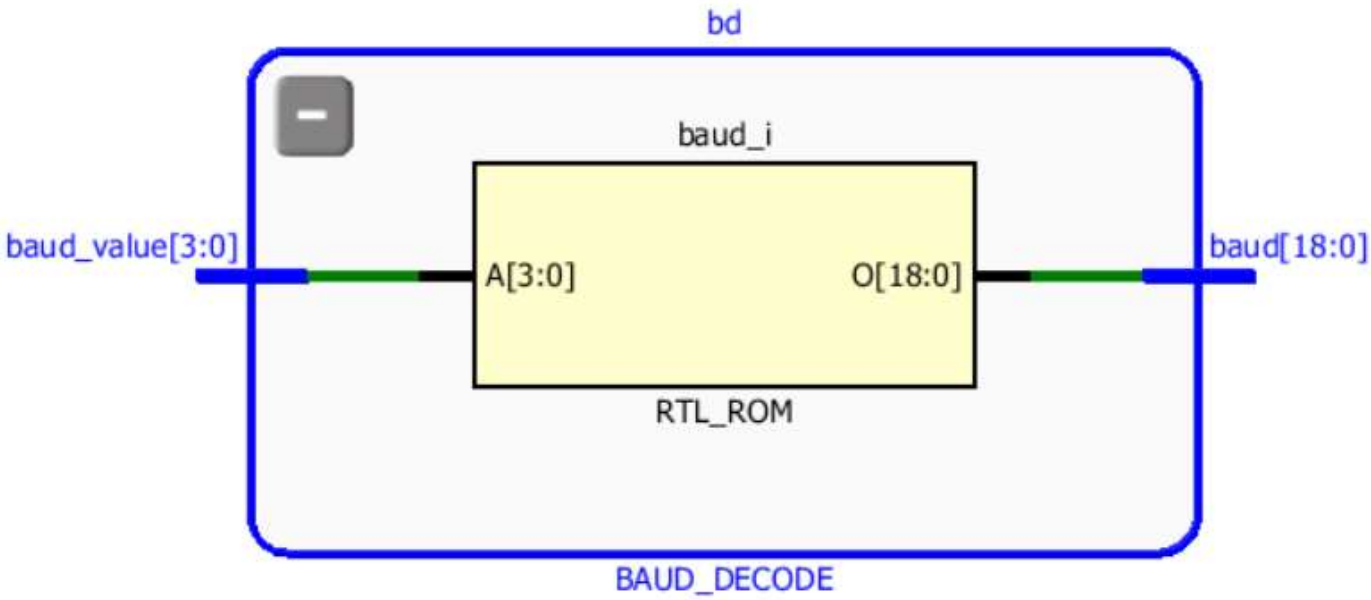

Figure 9: Baud Decode Block Diagram

10.3 I/O Table

Signal	Bits	I/O	Description
baud_value	4	Input	On board switches to select the baud rate
baud	19	Output	Outputted value for the selected baudrate

Table 10: Baud Decode I/O

11. LEDs Logic Design

11.1 Description

The LED design is a little extra feature on the Full UART where the LEDs will start from the beginning and slowly move to the left on the on the Nexys A7 onboard LEDs. Once the final LED reaches the LED[15], then it'll go back to the beginning of LED[0] and create a shifting sequence pattern for aesthetics; however, a more detailed conceptual approach will be explained further through this report on the assembly portion of it all. The small LED design did not receive its own module, but it is simply a loadable register. The load will be when the WRITE_STROBE and PORT_ID are both high depending on the assigned output designated for the LEDs on the .coe file. If reset is high, then the LEDs will output a '0', but if a load is asserted, then the LEDs will output the shifted value from the .coe file. More details about the logic about the LEDs will be shown in the assembly portion for the Transmit Engine design.

11.2 Block Diagram

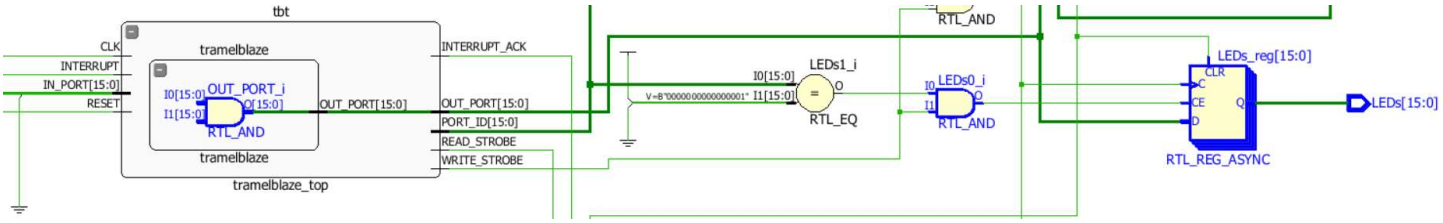

Figure 10: LEDs Block Diagram

11.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
resetNew	1	Input	Asynchronous reset with synchronous de-assertion
PORT_ID_wire	16	Input	PORT_ID_wire == 16'h0001 to output the LEDs
WRITE_STROBE_wire	1	Input	WRITE_STROBE_wire must be high to output the LEDs
OUT_PORT_wire	16	Input	The 16 bit output coming from the .coe file
LEDS	16	Output	Output for the onboard LEDs

Table 11: LEDs I/O

12. Transmit Engine Design

12.1 Description

The Transmit Engine module is in charge of transmitting the given 8 bit data serially. The Transmit Engine is made up of an SR Flop, an SSR flop, 2 bit counters, a decode module, and a shift register. The SR and SSR flops are used to set the 1 bit inputs to either '1' or '0'. The 2 bit counters are used to determine the time elapsed and the baud rate. The BTU counter will track the time it takes for a bit to send and the DONE counter will track the entire time it takes to send a message. The BTU counter is connected to the shift register and it will shift out a bit whenever the BTU counter reaches its maximum value.

12.2 Block Diagram

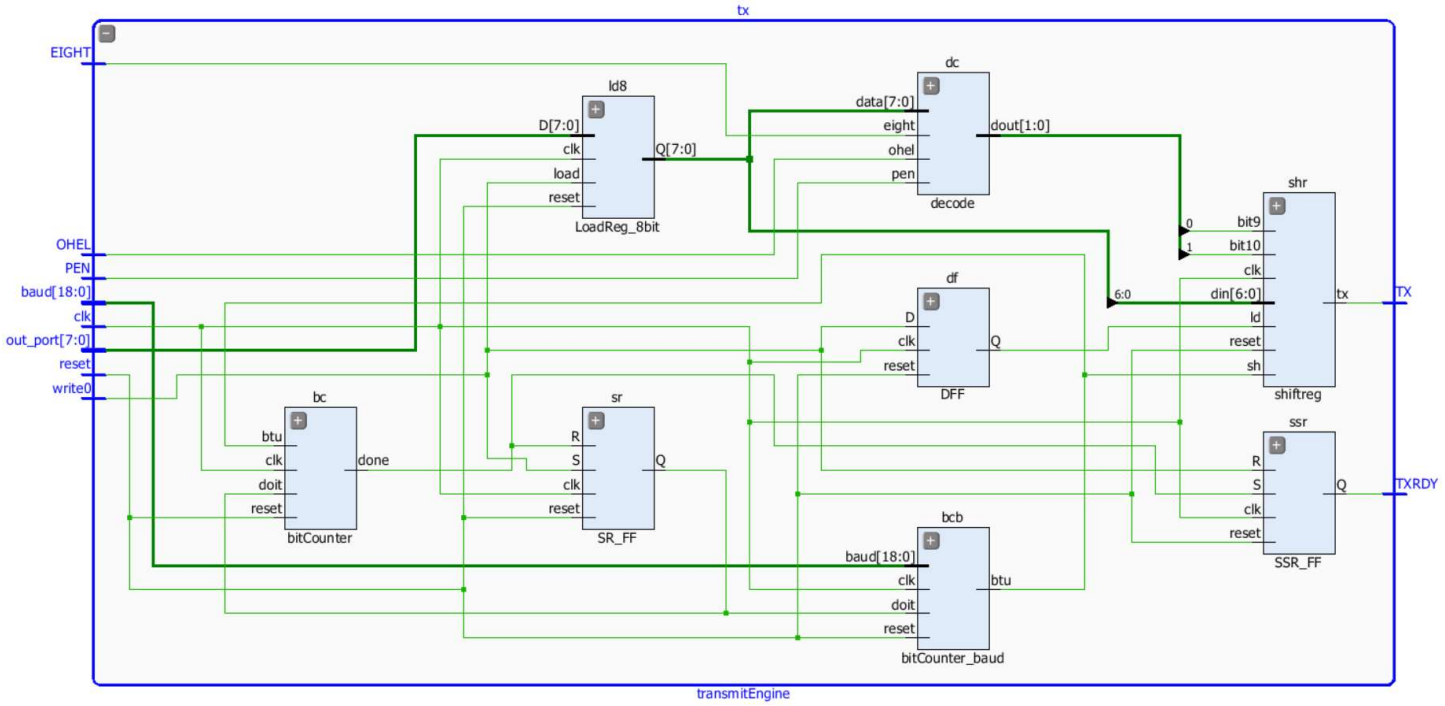

Figure 11: Transmit Engine Block Diagram

12.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion
EIGHT	1	Input	Eighth enable input to determine between 7/8 bits
PEN	1	Input	Parity enable input to determine if there is parity
OHEL	1	Input	Enable input to determine between even/odd parity
write0	1	Input	Load for the loadable register
baud	19	Input	Selected baud rate
out_port	8	Input	Outputted data from the .coe file
TXRDY	1	Output	Will output after the DONE counter reaches its max
TX	1	Output	Will output after the BTU counter reaches its max

Table 12: Transmit Engine I/O

13. SSR(Set/Set/Reset) Flip Flop Design

13.1 Description

The SSR Flop is very similar to the SR flop, but with one minor difference. The difference is located at the positive edge of the asynchronous reset with synchronous de-assertion. Instead of outputting a '0' when the reset signal is high, the SSR Flop will output a '1'.

13.2 Block Diagram

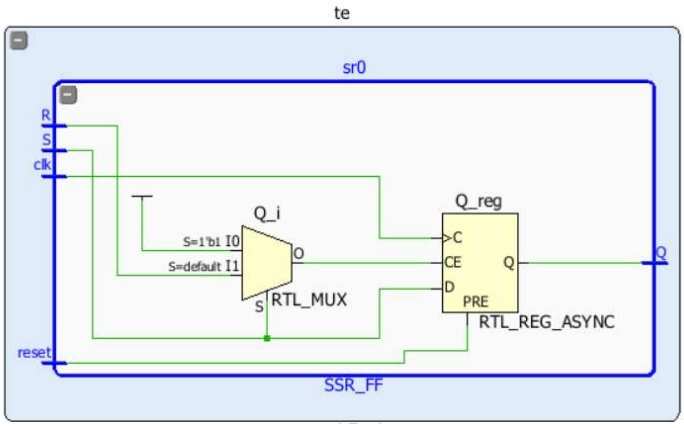

Figure 12: SSR Flop Block Diagram

13.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion(will output a '1')
S	1	Input	SET input of the SR flop
R	1	Input	RESET input of the SR flop
Q	1	Output	Output of the SR flop

Table 13: SSR Flop I/O

14. D(Data) Flip Flop Design

14.1 Description

A 1-bit memory cell to store binary data.

14.2 Block Diagram

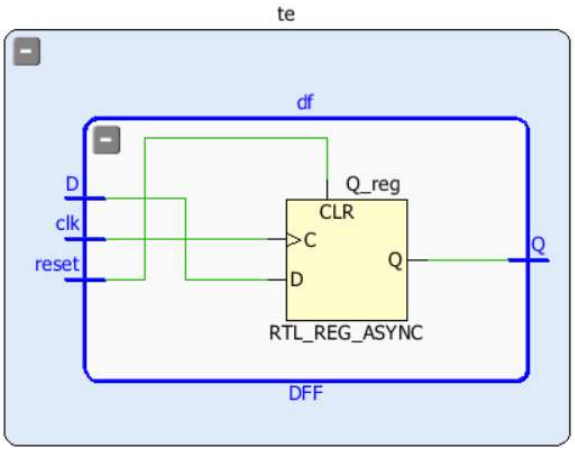

Figure 13: DFF Block Diagram

14.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion
D	1	Input	Input flop
Q	1	Output	Output flop

Table 14: DFF I/O

15. Bit Counter with Baud Design

15.1 Description

This first bit counter refers to it being the BTU bit counter. The reason that it is the BTU bit counter is because it is a simple counter, but it will output the BTU signal necessary to drive the select for the 4 to 1 mux. The BTU bit counter is also necessary because it is connected to the shift register. The BTU bit counter will allow the shift register to shift a bit every time the BTU counter reaches its maximum value. The maximum value is set depending on the preferable baud rate. The baud rate values were given by professor Tramel and were explained earlier through this Chip Spec.

15.2 Block Diagram

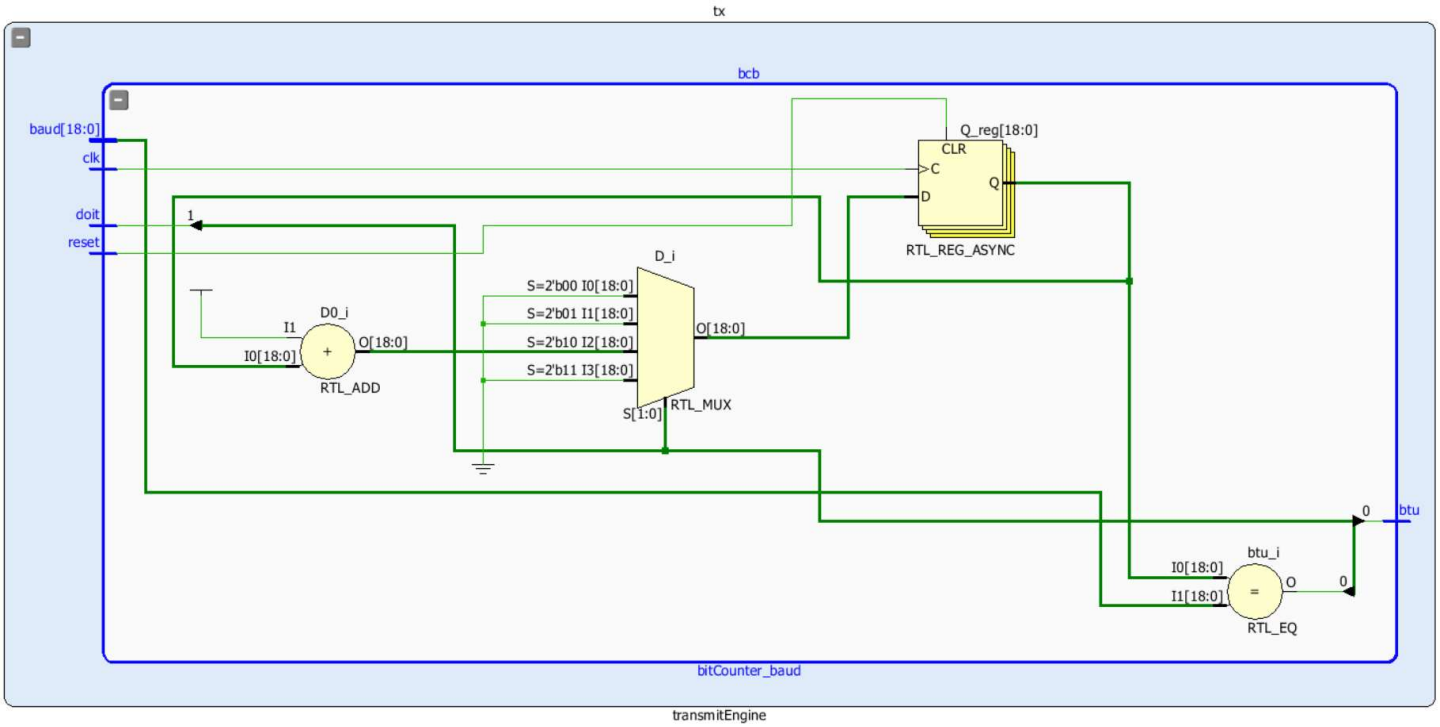

Figure 14: Bit Counter with Baud Block Diagram

15.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion
baud	19	Input	Baud rate selected from the onboard switches[7:4]
doit	1	Input	Is triggered high when data is outputted fom the .coe file
btu	1	Output	Will tell the shift register to shift when high

Table 15: Bit Counter with Baud I/O

16. Bit Counter Design

16.1 Description

This second bit counter is designed similarly to the other bit counter, but with different values for the multiplexor. This bit counter is primarily important because it keeps track of the entire time it takes to send a message. This module will output it directly to the SR flop inside the Transmit Engine and will allow TXRDY to output as high when DONE signal is high.

16.2 Block Diagram

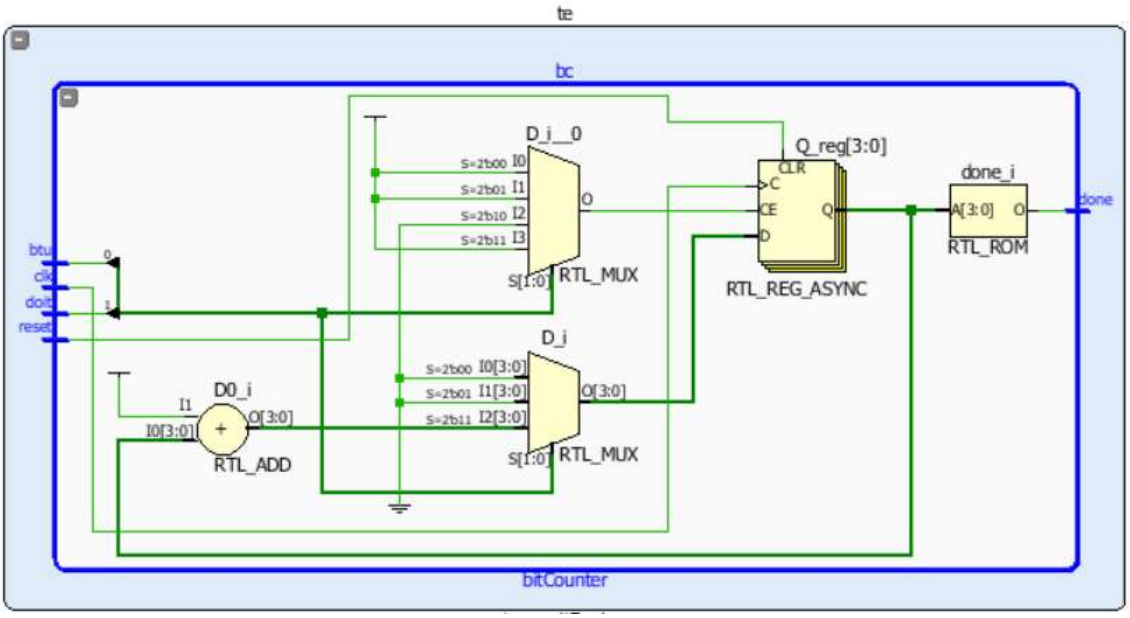

Figure 15: Bit Counter Block Diagram

16.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion
btu	1	Input	Is triggered when the BTU bit counter reaches maximum baud value
doit	1	Input	Is triggered high when data is outputted fom the .coe file
done	1	Output	Outputs high when a message has been processed fully

Table 16: Bit Counter I/O

17. Loadable Register 8 Bit Design

17.1 Description

This 8 bit loadable register is meant to hold the 8 bit data from the .coe file until a load is active or another signal sends out an enable bit to release the loaded data.

17.2 Block Diagram

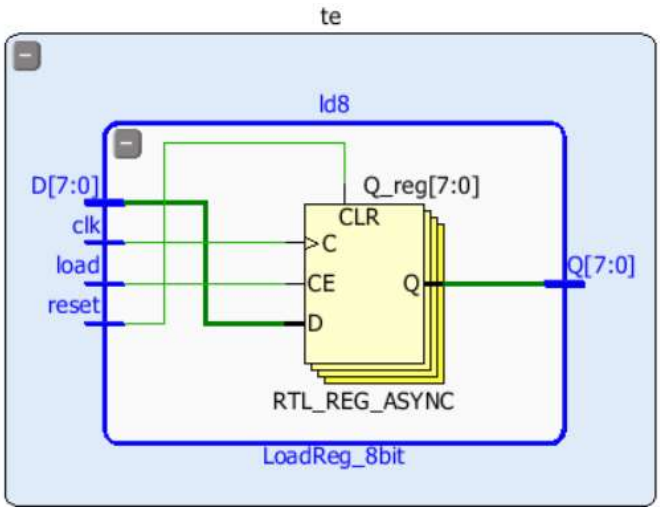

Figure 16: Loadable Register 8 Bit Block Diagram

17.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys A7 board
reset	1	Input	Asynchronous reset with synchronous de-assertion
load	1	Input	Load for the loadable register to output the 8 bit data from the .coe file
D	8	Input	The held 8 bit data
Q	8	Output	Output the 8 bit data

Table 17: Loadable Register 8 Bit I/O

18. Decode Design

18.1 Description

The decode module will take in the 8 bit data and check for any changes to the serial data based on three switch inputs such as 7/8 bits, parity enabled/disabled, or even/odd parity. Input EIGHT refers to whether the data is 7 or 8 bits. Input PEN refers to parity being enabled or disabled. Input OHEL refers to whether the parity is even or odd.

18.2 Block Diagram

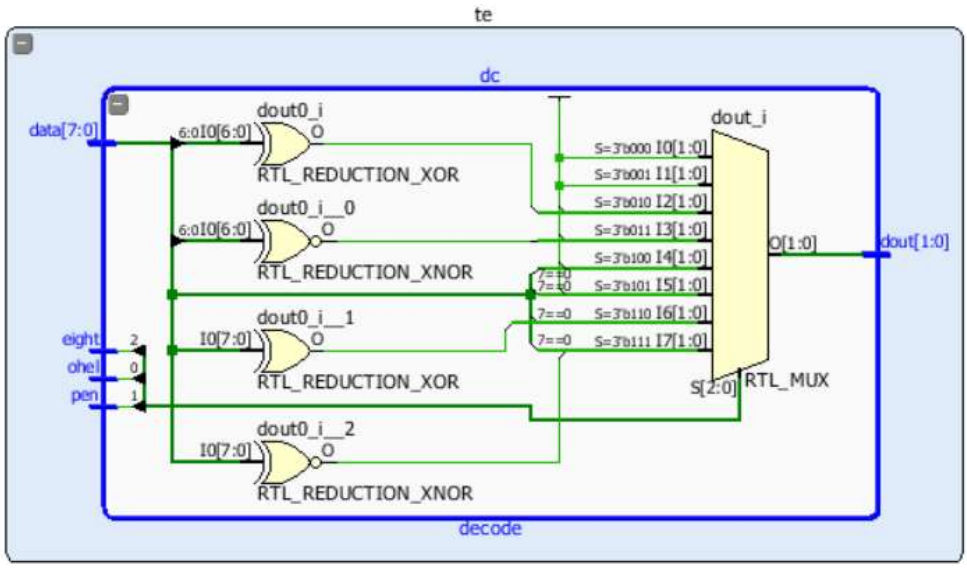

Figure 17: Decode Block Diagram

18.3 I/O Table

Signal	Bits	I/O	Description
eight	1	Input	Mapped to switch[3] to determine between 7 or 8 bits
pen	1	Input	Mapped to switch[2] to determine if parity is enabled or disabled
ohel	1	Input	Mapped to switch[1] to determine if parity is even or odd
data	8	Input	8 bit data from the .coe file
dout	2	Output	Output checked data to the shift registered

Table 18: Decode I/O

19. Shift Register Design

19.1 Description

The Shift Register is meant to shift the data serially. Many inputs from this design have come from modules that have already been explained, but here is a quick review. Inputs bit9 and bit10 come from the decode module after it checks the bits to determine if the data is 7/8 bits, if parity is enabled/disabled, or if the parity is even/odd. Input din comes from the loadable register and it will take 7 out of 8 bits from the loadable register. Input ld comes from the D(data) flip flop and the D(data) flip flop will hold the 1 bit memory of the load for the .coe message. Input sh comes from the BTU bit counter and it will be high when the BTU counter reaches the maximum value depending on the baud rate selected. Lastly, tx will be the least significant bit of the shifted serial data. For this design, the tx will have to be mapped to D4 in order for the serial terminal, RealTerm, to receive the serial data.

19.2 Block Diagram

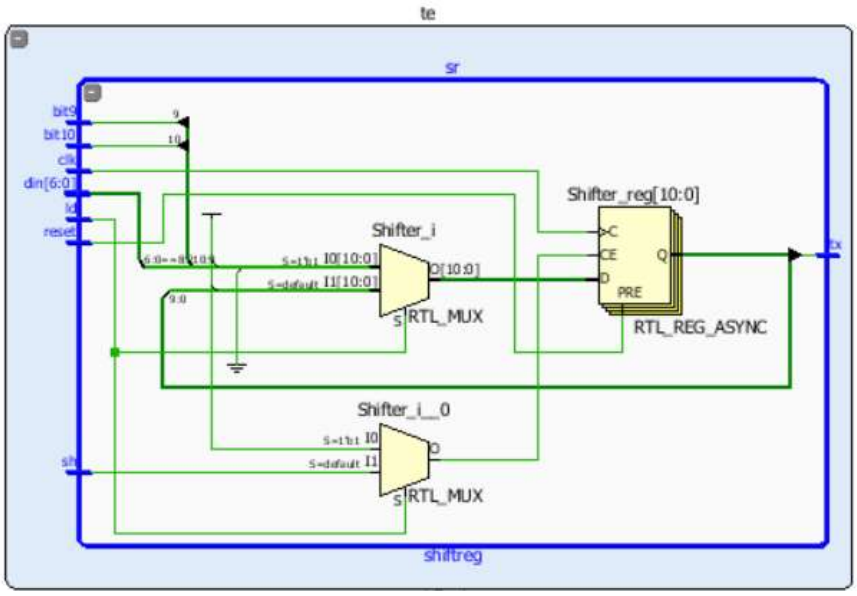

Figure 18: Shift Register Block Diagram

19.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys board
reset	1	Input	Asynchronous reset with synchronous de-assertion
bit10	1	Input	dout[1] from the decode module
bit9	1	Input	dout[0] from the decode module
din	7	Input	The least significant 7 bits from the loadable register
Ld	1	Input	Load for the loadable register to output the 8 bit data from the .coe file
Sh	1	Input	BTU counter output to shift data
Tx	1	Output	TX output that is mapped to D4

Table 19: Shift Register I/O

20. Receive Engine Design

20.1 Description

The Receive Engine module oversees receiving serial data. The Receive Engine is equipped with a finite state machine for receiving data and two counters that are similar to the Transmit Engine. One of those counters is a similar BTU counter from the Transmit Engine except that it will count for half a bit time for it can synchronize the data that has been captured. The other counter is similar to the DONE counter from the Transmit engine where it will count to either 9 or 11 bits depending on the EIGHT and PEN inputs. The Receive Engine is also equipped with error checking which checks for Framing, Overflow, and Parity Errors. The RX output is the 1 bit received data bit that will only output after the RX ready flag is high.

NOTE: Unlike the Transmit Engine, the Receive Engine was all made in 1 module; therefore, every block mentioned from the Receive Engine will be found in the RX_Engine.v at end of the Chip Spec in the appendix section.

20.2 Block Diagram

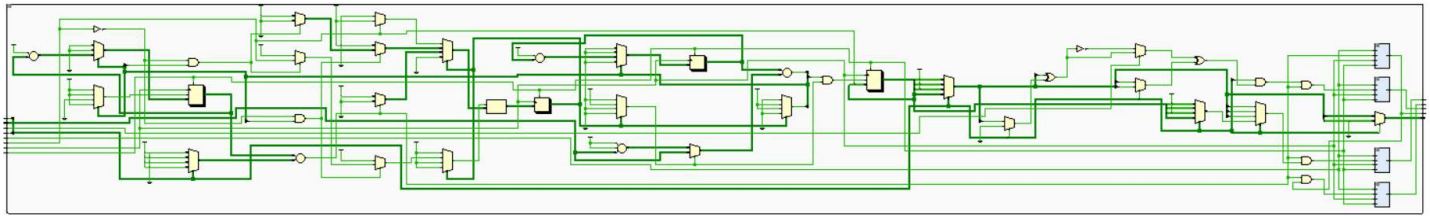

Figure 19: Receive Engine Block Diagram

20.3 I/O Table

Signal	Bits	I/O	Description
clk	1	Input	100MHz clock from the Nexys board
reset	1	Input	Asynchronous reset with synchronous de-assertion
RX	1	Input	A 1 bit receieved data bit.
EIGHT	1	Input	Eighth enable input to determine between 7/8 bits
PEN	1	Input	Parity enable input to determine if there is parity
OHEL	1	Input	Enable input to determine between even/odd parity
K	19	Input	The selected baud rate value
READS0	1	Input	READS[0] is the clear
UART_RDATA	8	Output	The 8 bit received data
RXRDY	1	Output	RX ready flag when ready to receive data
PERR	1	Output	Parity Error Flag
FERR	1	Output	Framing Error Flag
OVF	1	Output	Overflow Error Flag

Table 20: Receive Engine I/O

21. Receive Engine FSM Design

21.1 Description

The Receive Engine is equipped with a Finite State Machine (FSM) that will track the state of the control signals. The FSM keeps track of when the start bit is detected and will set the DOIT signal high while the data is being processed. The FSM will also check for when the data is finished being processed and the DONE counter will output DONE to be high which represents that all 11 bits have been received.

21.2 Figure

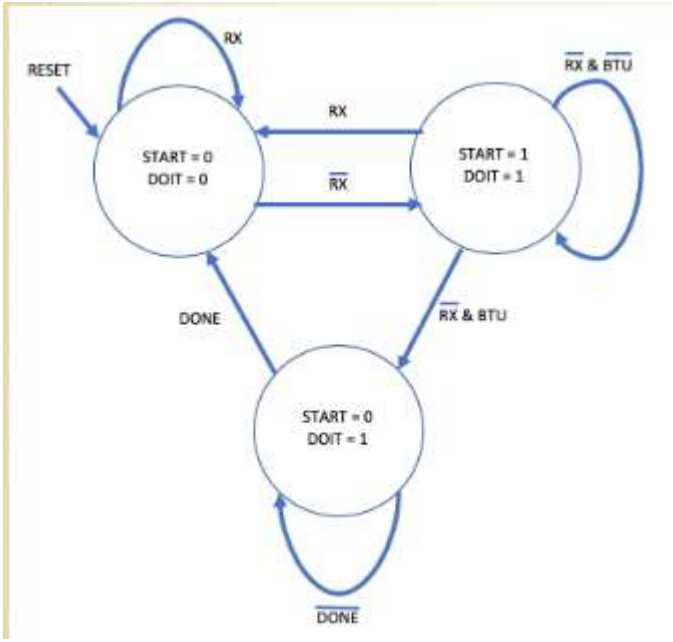

Figure 20: Receive Engine FSM Diagram

22. Receive Engine DONE Bit Counter Design

22.1 Description

The Receive Engine's DONE counter is very similar to the one that the Transmit Engine has. This DONE counter will count to either 9 or 11 bits depending on the EIGHT and PEN inputs.

22.2 Figure

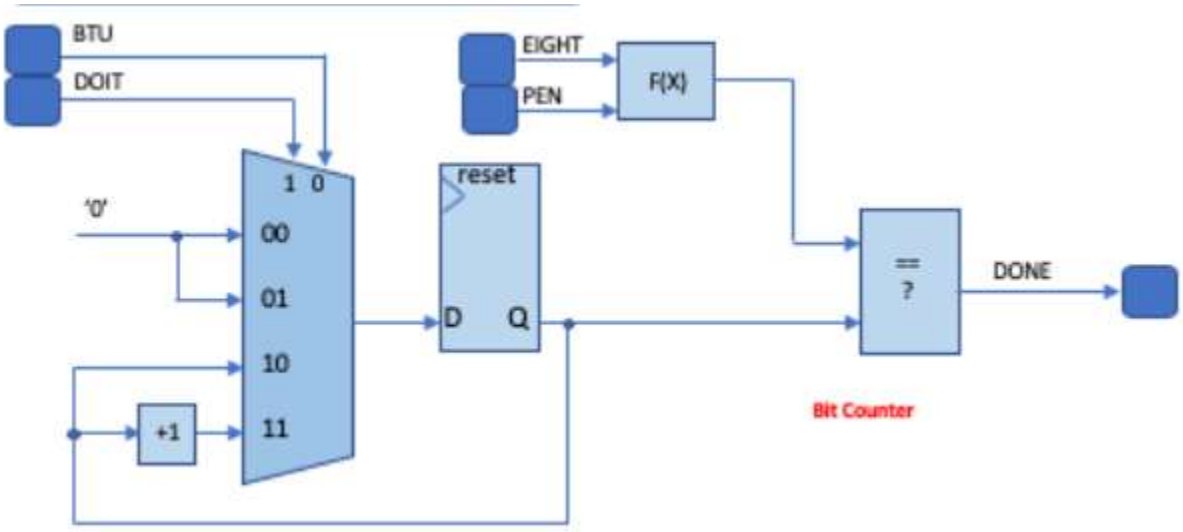

Figure 21: Receive Engine DONE Bit Counter Block Diagram

23. Receive Engine BTU Bit Counter Design

23.1 Description

The BTU counter from the Receive Engine is very similar to the BTU counter from the Transmit Engine. The BTU counter from the Receive Engine will count for half a bit time for it can synchronize the data that has been captured and then it will output a BTU signal.

23.2 Figure

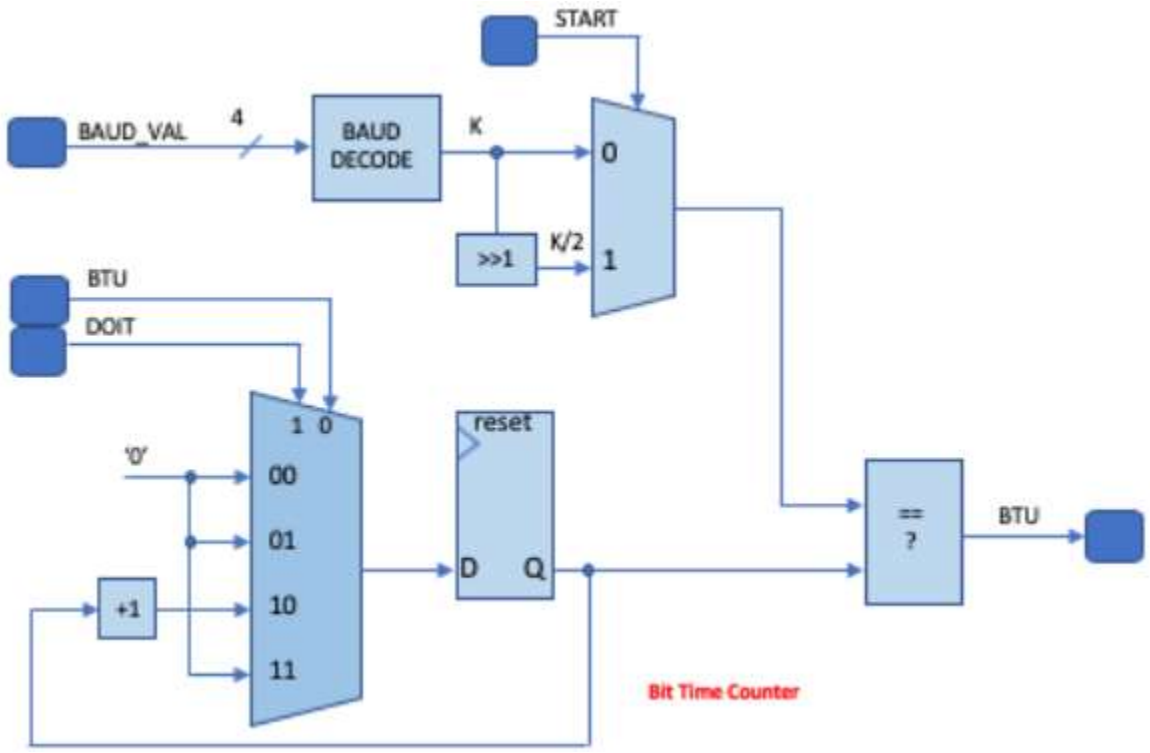

Figure 22: Receive Engine BTU Bit Counter Block Diagram

24. Receive Engine Shift Register Design

24.1 Description

The Receive Engine's Shift Register is designed to shift 10 bit data. The SHIFT is mapped to the BTU counter and if signal SHIFT is high, it will shift the data to the right to rid itself from the least significant bit (LSB).

24.2 Figure

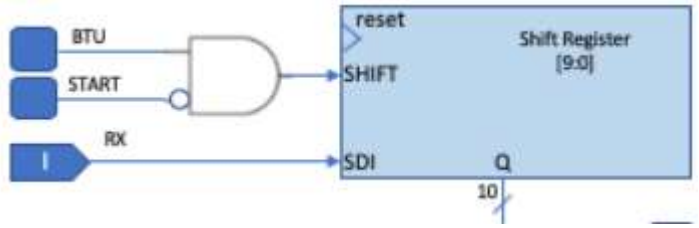

Figure 23: Receive Engine Shift Register Block Diagram

25. Receive Engine Right Justify Design

25.1 Description

The Right Justify block is designed to shift twice if EIGHT and PEN are low, but shift by 1 if either one of the 2 inputs is high; however, if both inputs are high, then the Right Justify block will not shift and the data remains the same. Also I did a quick tweak to this module to allow 7 bits to output as UART_RDATA instead of it only being 8 bits. The code is coded up to allow 8 bits to be outputted only if EIGHT input is high, but will output 7 bits if EIGHT input is low.

25.2 Figure

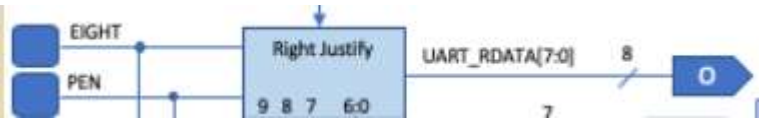

Figure 24: Receive Engine Right Justify Block Diagram

26. Receive Engine Status Design

26.1 Description

This next portion is mini logic that is made up of multiple multiplexors and SR flip flops. The figure underneath will show the routing it took to allow the rest of the blocks to flow for the Generated Parity Select, the Received Parity Select, and the Stop Bit Select. As well as error checking for the RX Ready flag, the Parity Error flag, the Framing Error flag, and the Overflow Error flag. Lastly, the Receive Engine is equipped with a clear option in a form of SR flip flops using the input of READS0.

26.2 Figure

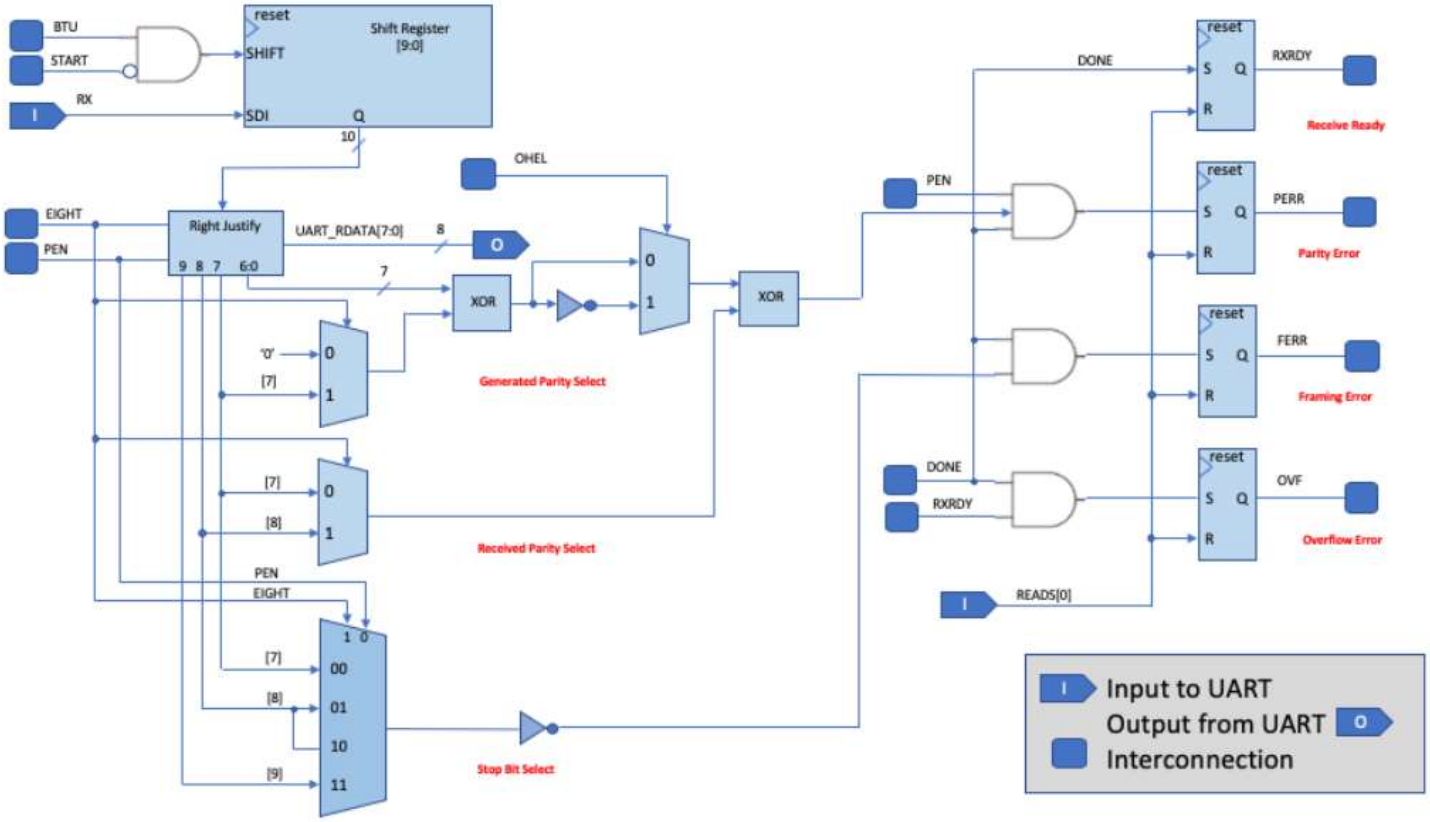

Figure 25: Receive Engine Status Block Diagram

27. Assembly Design

27.1 Description

For the assembly design to create the appropriate .coe file to run the UART to interface with the Transmit Engine, the Receive Engine, and the TramelBlaze processor. The assembly could be found at the appendix of the Chip Spec under the name of receiver.tba; however, the Bin to ASCII flowchart and the FIND IT flowchart can be found underneath to show the conversion from a binary value to an ASCII character.

27.2 BIN to ASCII & Find It Flowchart

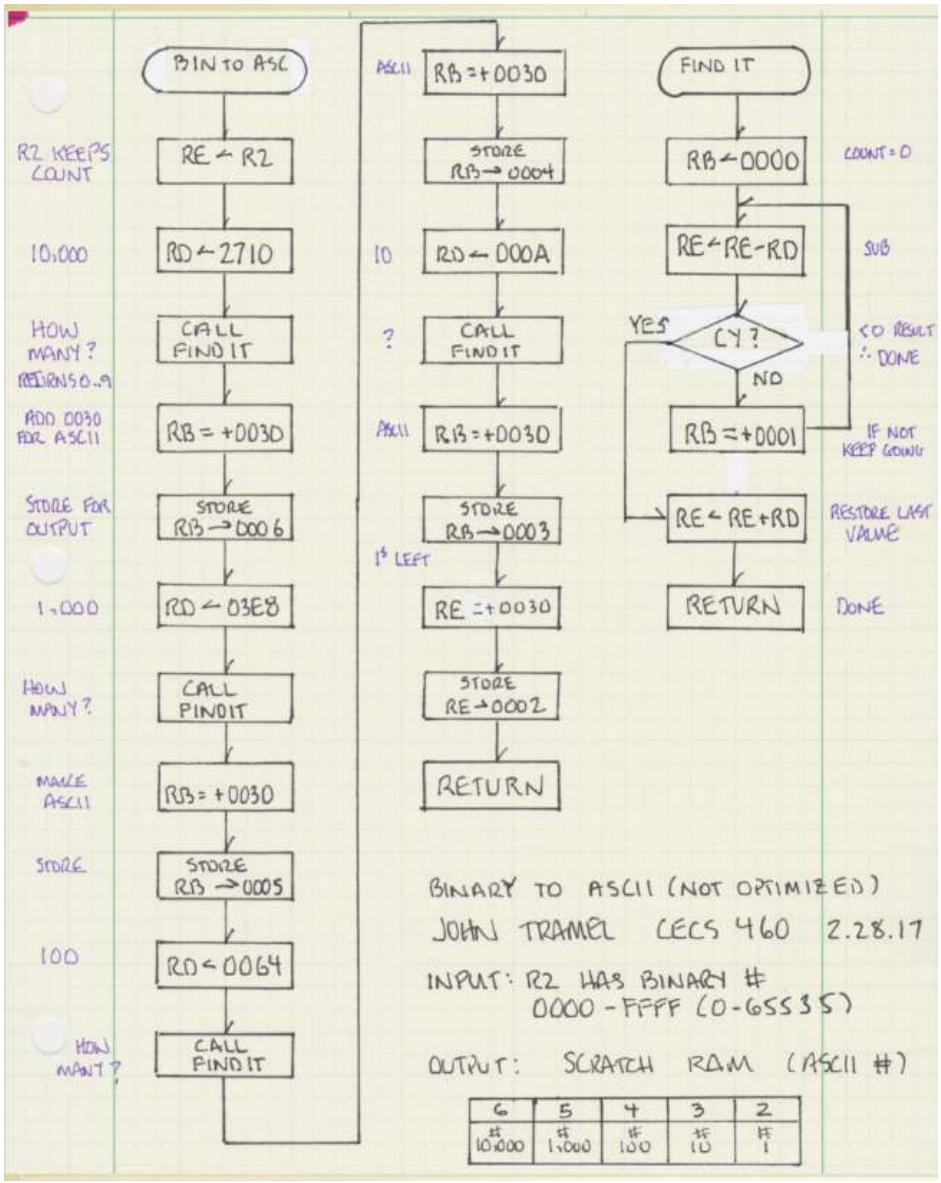

Figure 26: BINTOASCCI & FINDIT Flowchart

27.3 I/O Table

Output	Value
RXRDY	0000
TXRDY	0000
ISR	0001
LEDs	0001

Table 21: PORT_ID Output Values

28. Transmit Engine Simulation

28.1 Description

This simulation is to test a quick pulse of when write0, the load, is high for 100ns in the Transmit Engine. The baud rate was set to the maximum baud value and many input combinations for EIGHT, PEN, and OHEL were tested to make sure that the Transmit Engine is working properly.

28.2 Simulation

Figure 27: Transmit Engine Simulation

For this simulation, the input switches are set high, the baud value is set to the highest baud rate, and out_port value is set to 0x07 for testing. This simulation was not necessary, but it was meant to check if every module is functioning properly by following all the modules to see if the transmit engine is outputting properly.

29. Receive Engine Simulation

29.1 Description

This simulation is to test that the data will receive the hex value of 0xAA. The test bench is found at the appendix under the name of RX_Engine_tb.v, and it shows how each RX input is pressed to shift the correct data. Since the data is being shifted in a form of first in first out (FIFO). Notice where the START bit begins and where the DOIT bits continue all the way until the end after receiving 0xAA. Also notice how the bits are shifting depending on the value of RX. Lastly notice where RXRDY is high when the data is done being shifted to get 0xAA. That will signal that the data has been received.

29.2 Simulation

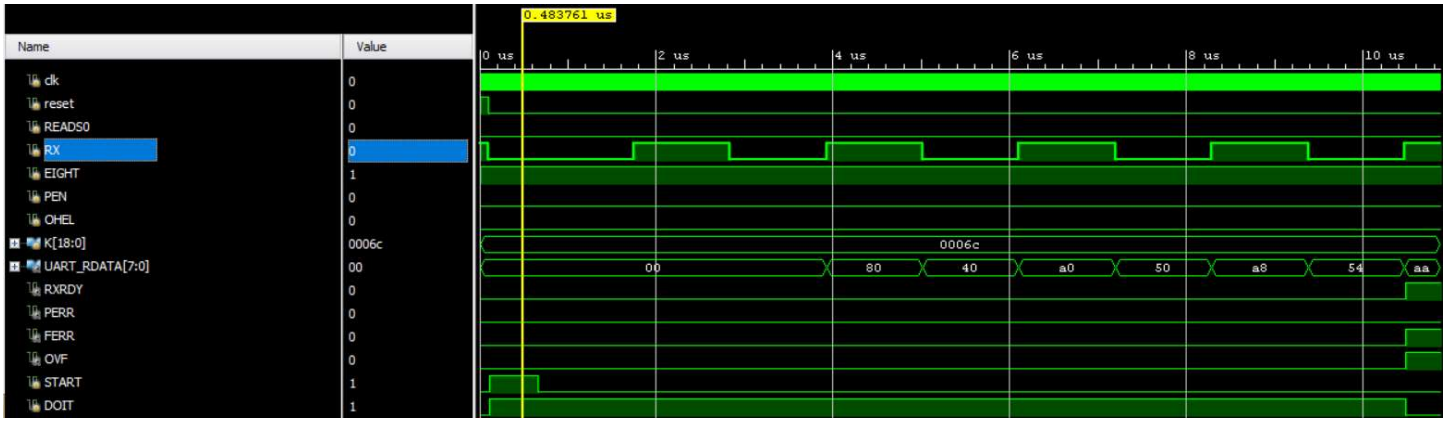

Figure 28: Receive Engine Simulation

30. Full UART Simulation

30.1 Description

This simulation is to test a if the prompt banner and the prompt was being outputted. In the figure underneath, notice how ‘F’ is being outputted. That ‘F’ represents the first letter of the prompt which is “FRANCO UART”. Unfortunately, the reset of the figures cannot be seen, but under the Full UART Verification section of this Chip Spec, there can be figures to show the functionality of this project.

30.2 Simulation

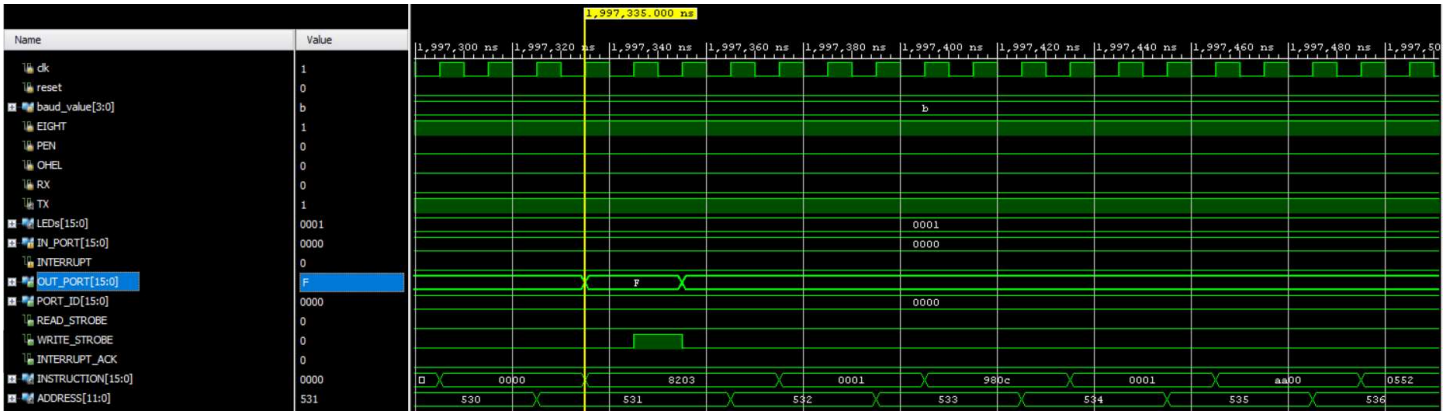

Figure 29: Full UART Simulation

31. Full UART Verification

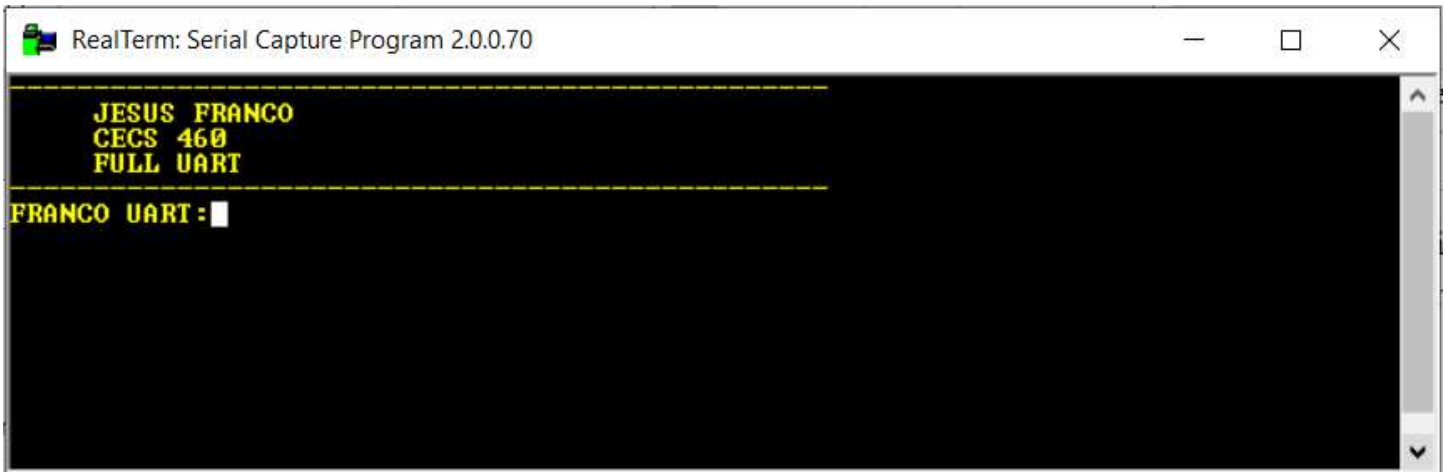

Figure 30: Full UART Verification 1

This figure above shows the banner and the prompt for the Full UART project.

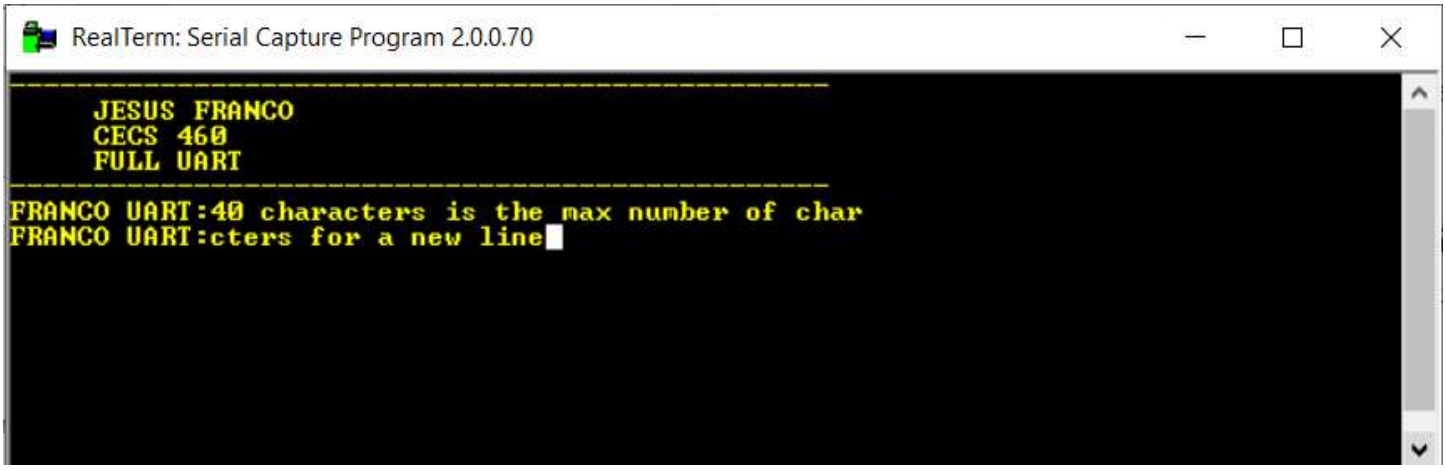

Figure 31: Full UART Verification 2

This next figure shows the max characters to be 40 in a single line before switching into the next line.

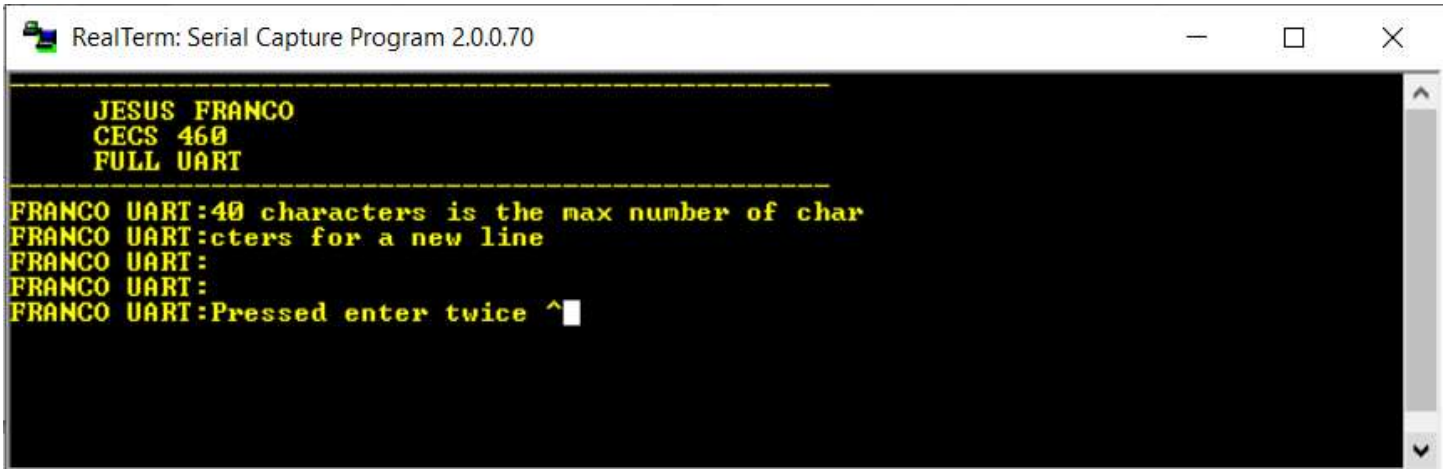

Figure 32: Full UART Verification 3

This next figure shows the ENTER button being pressed twice.

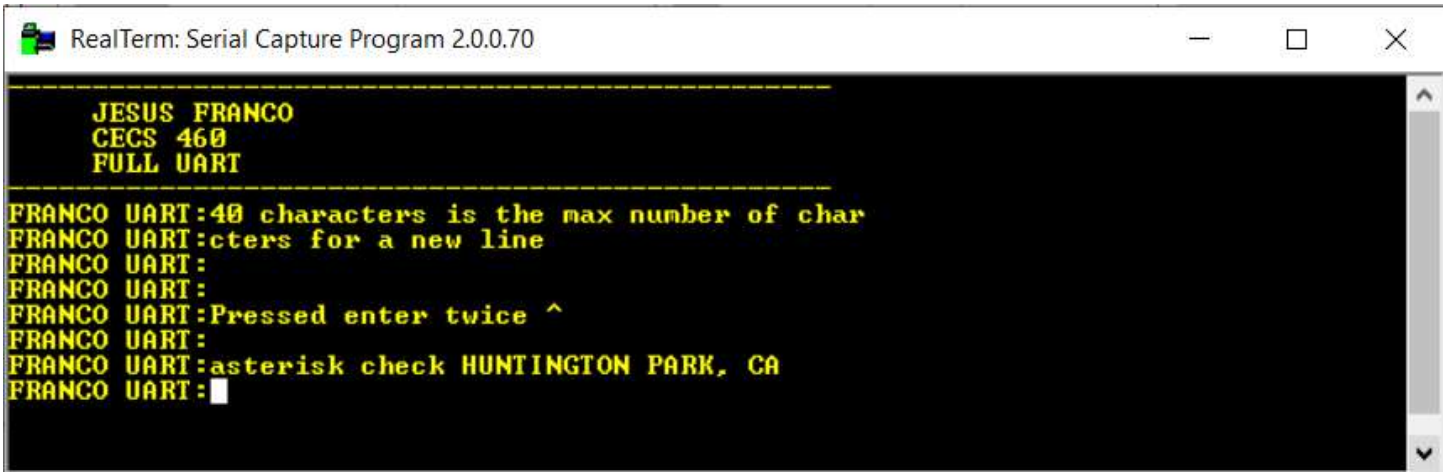

Figure 33: Full UART Verification 4

This next figure shows the * character being pressed to show the hometown.

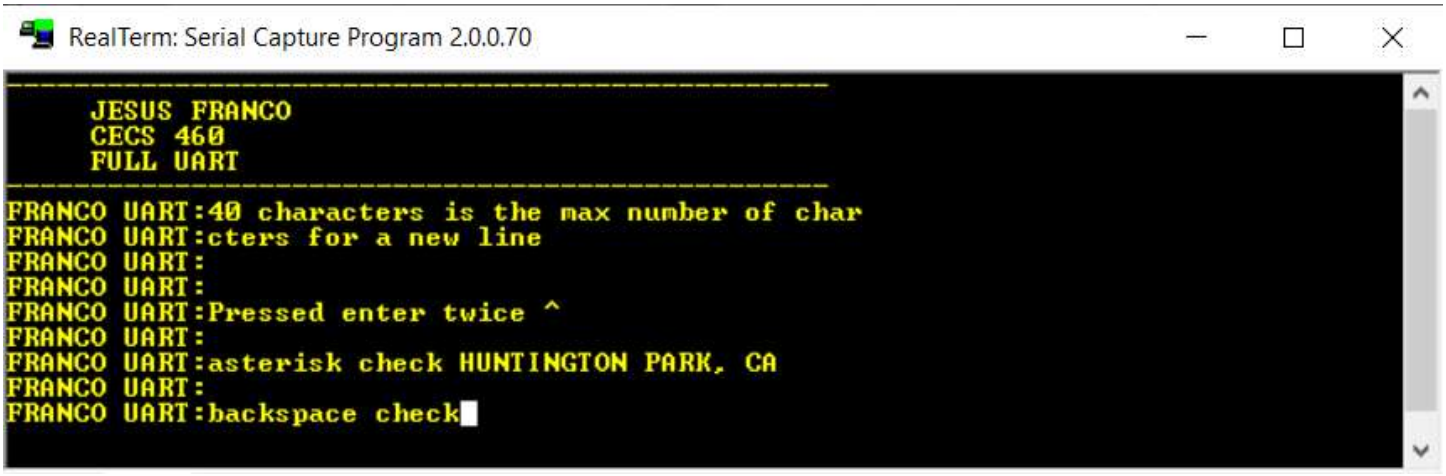

Figure 34: Full UART Verification 5

This next figure shows before the BACKSPACE button is pressed.

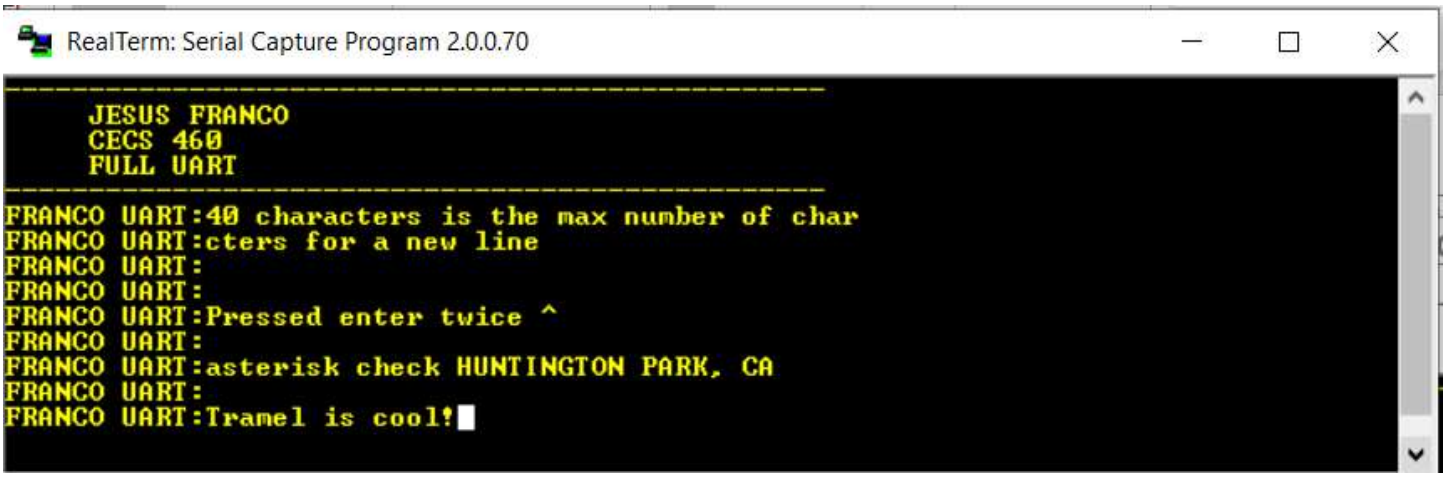

Figure 35: Full UART Verification 6

This next figure shows after the BACKSPACE button has been pressed.

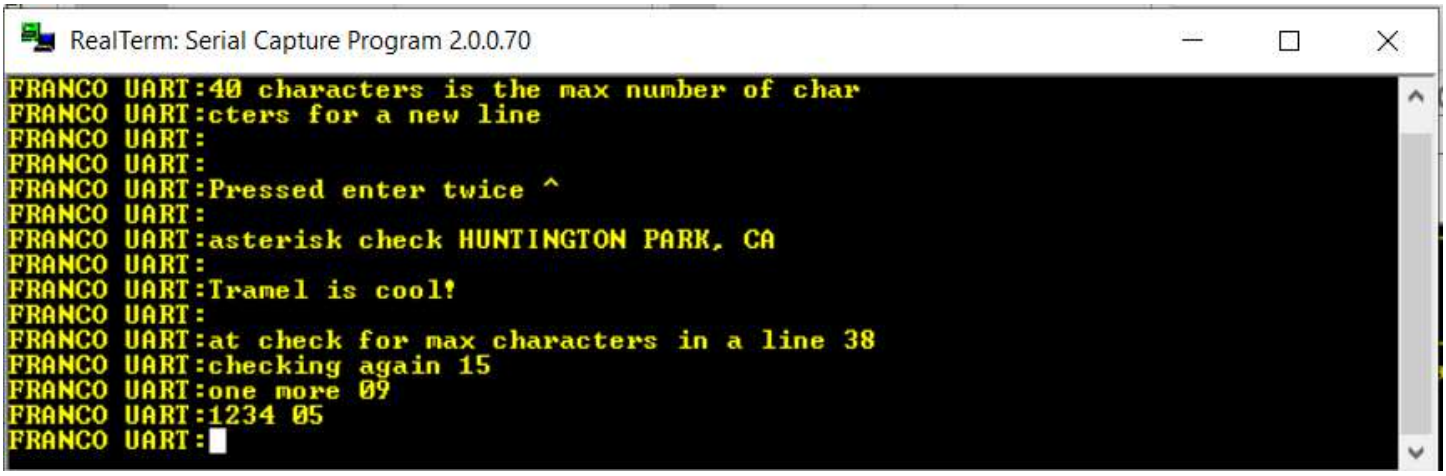

Figure 36: Full UART Verification 7

Lastly, this next figure will show the number of characters count that is being recorded by pressing the @ symbol. The counter will keep track of the characters on each line and will be set to 0 on reset.

32. Constraints File

32.1 Description

The constraints file is not necessary, but there is one minor detail that should be explained; therefore, for completeness, the full constraints file will be found at the appendix for this Chip Spec. The only minor detail that requires an explanation is the TX and RX output from the design. The packaged pin for TX is supposed to be mapped directly to C4; however, TX needs to be mapped directly to D4 and RX needs to be mapped to C4 so the serial terminal, RealTerm, could output the results for serial communication between the Nexys A7 board and RealTerm.

32.2 Figure

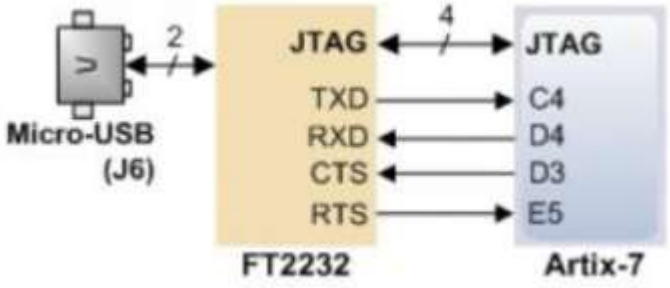

Figure 37: USB-UART Bridge (Serial Port)

33. Appendix

33.1 receiver.tba

```
//*************************************************************//
// File name: receiver.tba                                     //
//                                                             //
// Created by       Jesus Franco                               //
// Copyright © 2020 Jesus Franco on 11/10/2020 08:08:31 PM     //
//                                                             //
//                                                             //
// In submitting this file for class work at CSULB             //
// I am confirming that this is my work and the work           //
// of no one else. In submitting this code I acknowledge that  //
// plagiarism in student project work is subject to dismissal. //
// from the class                                              //
// Dependencies:                                               //
//*************************************************************//

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;DECLARATIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ASCII_BS        EQU   0008  ; Back Space
ASCII_LF        EQU   000A  ; <LF> Line Feed
ASCII_CR        EQU   000D  ; <CR> Carriage return
ASCII_aster     EQU   002A  ; '*'  asterisk
ASCII_AT        EQU   0040  ; '@'  at
ASCII_DOT       EQU   002E  ; '.'  period
ASCII_EQ        EQU   003D  ; '='  equal
ASCII_DASH      EQU   002D  ; '-'  dash
ASCII_COL       EQU   003A  ; ':'  colon
ASCII_SP        EQU   0020  ; ' '  space
ASCII_comma     EQU   002C  ; ','  comma
ASCII_0         EQU   0030  ; '0'  zero
ASCII_1         EQU   0031  ; '1'  one
ASCII_2         EQU   0032  ; '2'  two
ASCII_3         EQU   0033  ; '3'  three
ASCII_4         EQU   0034  ; '4'  four
ASCII_5         EQU   0035  ; '5'  five
ASCII_6         EQU   0036  ; '6'  six
ASCII_7         EQU   0037  ; '7'  seven
ASCII_8         EQU   0038  ; '8'  eight
ASCII_9         EQU   0039  ; '9'  nine
ASCII_A         EQU   0041  ; 'A'
ASCII_B         EQU   0042  ; 'B'
ASCII_C         EQU   0043  ; 'C'
ASCII_D         EQU   0044  ; 'D'
ASCII_E         EQU   0045  ; 'E'
ASCII_F         EQU   0046  ; 'F'
ASCII_G         EQU   0047  ; 'G'
ASCII_H         EQU   0048  ; 'H'
ASCII_I         EQU   0049  ; 'I'
ASCII_J         EQU   004A  ; 'J'
ASCII_K         EQU   004B  ; 'K'
ASCII_L         EQU   004C  ; 'L'
ASCII_M         EQU   004D  ; 'M'
```

```
ASCII_N         EQU   004E  ; 'N'
ASCII_O         EQU   004F  ; 'O'
ASCII_P         EQU   0050  ; 'P'
ASCII_Q         EQU   0051  ; 'Q'
ASCII_R         EQU   0052  ; 'R'
ASCII_S         EQU   0053  ; 'S'
ASCII_T         EQU   0054  ; 'T'
ASCII_U         EQU   0055  ; 'U'
ASCII_V         EQU   0056  ; 'V'
ASCII_W         EQU   0057  ; 'W'
ASCII_X         EQU   0058  ; 'X'
ASCII_Y         EQU   0059  ; 'Y'
ASCII_Z         EQU   005A  ; 'Z'
forty           EQU   0028  ; hex value for 40

start_banner    EQU     0000  ; banner start counter
end_banner      EQU     0098  ; banner end counter

start_prompt    EQU     0098  ; prompt start counter
end_prompt      EQU     00A4  ; prompt end counter

start_hometown  EQU     00A4  ; hometown start counter
end_hometown    EQU     00BD  ; hometown end counter

start_bs        EQU     00BD  ; backspace start counter
end_bs          EQU     00C0  ; backspace end counter

start_crlf      EQU     00C0  ; crlf start counter
end_crlf        EQU     00C2  ; crlf end counter

COUNT_BEGIN     EQU     00C2  ; count begin
COUNT_TEN       EQU     00C2  ; count ten
COUNT_ONE       EQU     00C3  ; count one
COUNT_END       EQU     00C4  ; count end


TEMP            EQU     R0
POINTER         EQU     R3
STATUS          EQU     R1
DATA            EQU     R2
LEDS            EQU     R4

CASE   EQU RC

CHAR_COUNT EQU RD

DELAY_COUNT1    EQU     R5
DELAY_COUNT2    EQU     R6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INITIALIZATION CODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Init
        LOAD    TEMP,           0000
        LOAD    POINTER,        0000
        LOAD    CHAR_COUNT,     0000
        LOAD    STATUS,         0000
```

```
        LOAD    DATA,           0000
        ; case is initialized to 1 because
        ; the banner is to print at startup
        LOAD    CASE,           0001


        LOAD    DELAY_COUNT1,   0000
        LOAD    DELAY_COUNT2    0000
        LOAD    LEDS,           0001

        CALL    initialize_banner   ; load the necessary chars for header in scratchRam
        CALL    initialize_prompt   ; load necessary chars for prompt
        CALL    initialize_hometown
        CALL    initialize_backspace
        CALL    initialize_crlf

        LOAD    POINTER, 0000           ; reset the pointer to 0

        ENINT


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MAIN SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Main
        OUTPUT  LEDS, 0001      ; output to port 0001
        RL      LEDS            ; shift LED left
        CALL    Delay           ; wait .01
        CALL    Delay           ; wait .01
        JUMP    Main

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;DELAY SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Delay
        ADD     DELAY_COUNT1, 0001   ; add 1 to DELAY_COUNT1
        ADDC    DELAY_COUNT2, 0000   ; DELAY_COUNT1 rolls from FFFF -> 0000
        COMP    DELAY_COUNT2, 000F   ; check if at the end of delay
        JUMPNZ  Delay                ; if not -> keep counting
        LOAD    DELAY_COUNT1, 0000   ; else -> reset counts and return
        LOAD    DELAY_COUNT2, 0000
        RETURN

        ADDRESS 0300

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ISR SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ISR
        INPUT   STATUS, 0001
        AND     STATUS, 0003

        COMP    STATUS, 0003
        JUMPZ   GOT_BOTH

        COMP    STATUS, 0002        ; check if TXRDY is high
        CALLZ   GOT_TXRDY
```

```
        COMP    STATUS, 0001        ; check if RXRDY is high
        CALLZ   GOT_RXRDY

        RETEN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GOT BOTH SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GOT_BOTH

        CALL GOT_TXRDY
        CALL GOT_RXRDY
        RETEN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;BINARY_TO_ASCII SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
BINARY_TO_ASCII
        LOAD    RE, CHAR_COUNT    ; load RE with counter


        LOAD    RD, 000A       ; RD <- 10
        CALL    FIND_IT        ; Call find it to set *RB (number of 10s)
        ADD     RB, 0030       ; add 0x0030 for ASCII conversion
        STORE   RB, COUNT_TEN       ; store

        ; 1's left
        ADD     RE, 0030       ; add 0x0030 for ASCII conversion
        STORE   RE, COUNT_ONE       ; store


        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FIND_IT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
FIND_IT
        LOAD    RB, 0000

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SUBTRACT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SUBTRACT
        SUB     RE, RD
        JUMPC   RESTORE_LAST
        ADD     RB, 0001
        JUMP    SUBTRACT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RESTORE_LAST SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
RESTORE_LAST
        ADD     RE, RD
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;INITALIZE BANNER SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
initialize_banner
        LOAD    TEMP, ASCII_DASH
x50     STORE   TEMP, POINTER
        ADD     POINTER, 0001       ; POINTER++
        ADD     CHAR_COUNT, 0001    ; COUNT++
        COMP    CHAR_COUNT, 0031    ; count == 49?
        JUMPC   x50
        LOAD    CHAR_COUNT, 0000

        COMP    POINTER, 0064
        JUMPNC  finish_banner

        LOAD    TEMP, ASCII_CR
        STORE   TEMP, POINTER       ; 50
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_LF
        STORE   TEMP, POINTER       ; 51
        ADD     POINTER, 0001

        ; 5 spaces
        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER       ; 52
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER       ; 53
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER       ; 54
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER       ; 55
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER       ; 56
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_J
        STORE   TEMP, POINTER       ; 57
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_E
        STORE   TEMP, POINTER       ; 58
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_S
        STORE   TEMP, POINTER       ; 59
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_U
        STORE   TEMP, POINTER       ; 60
```

```
ADD     POINTER, 0001

LOAD    TEMP, ASCII_S
STORE   TEMP, POINTER       ; 61
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 62
ADD     POINTER, 0001

LOAD    TEMP, ASCII_F
STORE   TEMP, POINTER       ; 63
ADD     POINTER, 0001

LOAD    TEMP, ASCII_R
STORE   TEMP, POINTER       ; 64
ADD     POINTER, 0001

LOAD    TEMP, ASCII_A
STORE   TEMP, POINTER       ; 65
ADD     POINTER, 0001

LOAD    TEMP, ASCII_N
STORE   TEMP, POINTER       ; 66
ADD     POINTER, 0001

LOAD    TEMP, ASCII_C
STORE   TEMP, POINTER       ; 67
ADD     POINTER, 0001

LOAD    TEMP, ASCII_O
STORE   TEMP, POINTER       ; 68
ADD     POINTER, 0001

LOAD    TEMP, ASCII_CR
STORE   TEMP, POINTER       ; 69
ADD     POINTER, 0001

LOAD    TEMP, ASCII_LF
STORE   TEMP, POINTER       ; 70
ADD     POINTER, 0001

; 5 spaces
LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 71
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 72
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 73
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 74
```

```
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 75
ADD     POINTER, 0001

LOAD    TEMP, ASCII_C
STORE   TEMP, POINTER       ; 76
ADD     POINTER, 0001

LOAD    TEMP, ASCII_E
STORE   TEMP, POINTER       ; 77
ADD     POINTER, 0001

LOAD    TEMP, ASCII_C
STORE   TEMP, POINTER       ; 78
ADD     POINTER, 0001

LOAD    TEMP, ASCII_S
STORE   TEMP, POINTER       ; 79
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 80
ADD     POINTER, 0001

LOAD    TEMP, ASCII_4
STORE   TEMP, POINTER       ; 81
ADD     POINTER, 0001

LOAD    TEMP, ASCII_6
STORE   TEMP, POINTER       ; 82
ADD     POINTER, 0001

LOAD    TEMP, ASCII_0
STORE   TEMP, POINTER       ; 83
ADD     POINTER, 0001

LOAD    TEMP, ASCII_CR
STORE   TEMP, POINTER       ; 84
ADD     POINTER, 0001

LOAD    TEMP, ASCII_LF
STORE   TEMP, POINTER       ; 85
ADD     POINTER, 0001

; 5 spaces
LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 86
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 87
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 88
```

```
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 89
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 90
ADD     POINTER, 0001

LOAD    TEMP, ASCII_F
STORE   TEMP, POINTER       ; 91
ADD     POINTER, 0001

LOAD    TEMP, ASCII_U
STORE   TEMP, POINTER       ; 92
ADD     POINTER, 0001

LOAD    TEMP, ASCII_L
STORE   TEMP, POINTER       ; 93
ADD     POINTER, 0001

LOAD    TEMP, ASCII_L
STORE   TEMP, POINTER       ; 94
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP
STORE   TEMP, POINTER       ; 95
ADD     POINTER, 0001

LOAD    TEMP, ASCII_U
STORE   TEMP, POINTER       ; 96
ADD     POINTER, 0001

LOAD    TEMP, ASCII_A
STORE   TEMP, POINTER       ; 97
ADD     POINTER, 0001

LOAD    TEMP, ASCII_R
STORE   TEMP, POINTER       ; 98
ADD     POINTER, 0001

LOAD    TEMP, ASCII_T
STORE   TEMP, POINTER       ; 99
ADD     POINTER, 0001

LOAD    TEMP, ASCII_CR
STORE   TEMP, POINTER       ; 100
ADD     POINTER, 0001

LOAD    TEMP, ASCII_LF
STORE   TEMP, POINTER       ; 101
ADD     POINTER, 0001

LOAD    TEMP, ASCII_DASH    ; 150
JUMP    x50
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FINNISH BANNER SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
finish_banner

        LOAD    TEMP, ASCII_CR
        STORE   TEMP, POINTER       ; 151
        ADD     POINTER, 0001
        LOAD    TEMP, ASCII_LF      ; 152
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    CHAR_COUNT, 0000

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INITIALIZE PROMPT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
initialize_prompt

        LOAD    TEMP, ASCII_F       ; 154
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_R       ; 155
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_A       ; 156
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_N       ; 157
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_C       ; 158
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_O      ; 159
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP       ; 160
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_U      ; 161
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_A       ; 162
        STORE   TEMP, POINTER
        ADD     POINTER, 0001
```

```
        LOAD    TEMP, ASCII_R       ; 163
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_T       ; 164
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_COL     ; 165
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INITIALIZE HOMETOWN SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
initialize_hometown

        LOAD    TEMP, ASCII_H       ; 164
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_U       ; 165
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_N      ; 166
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_T       ; 167
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_I       ; 168
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_N       ; 169
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_G       ; 170
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_T       ; 171
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_O       ; 172
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_N       ; 173
        STORE   TEMP, POINTER
```

```
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP       ; 174
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_P      ; 175
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_A       ; 176
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_R       ; 177
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_K      ; 178
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_comma       ; 179
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP       ; 180
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_C       ; 181
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_A       ; 182
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP   ; 183
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP      ; 184
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP       ; 185
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_SP      ; 186
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_CR      ; 187
STORE   TEMP, POINTER
ADD     POINTER, 0001
```

```
        LOAD    TEMP, ASCII_LF      ; 188
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INITALIZE BACKSPACE SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
initialize_backspace
        LOAD    TEMP, ASCII_BS      ; 191
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_SP      ; 192
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_BS      ; 193
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;INTIALIZE CRLF SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
initialize_crlf

        LOAD    TEMP, ASCII_CR      ; 194
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        LOAD    TEMP, ASCII_LF      ; 195
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GOT_TXRDY SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GOT_TXRDY
        COMP    CASE, 0000          ; if the case is 0, then return
        RETURNZ

        FETCH   TEMP, POINTER       ; fetch what's pointed to by the pointer
        OUTPUT  TEMP, 0000          ; output the data
        ADD     POINTER 0001        ; increment the pointer

        COMP    CASE, 0001          ; case 1 -> display the banner
        JUMPZ   PRINT_BANNER

        COMP    CASE, 0002          ; case 2 -> display the prompt
        JUMPZ   PRINT_PROMPT
```

```
        COMP    CASE, 0003
        JUMPZ   PRINT_HOMETOWN      ; case 3 -> display the hometown

        COMP    CASE, 0004          ; case 4 -> display the bs (bs sp bs)
        JUMPZ   PRINT_BS

        COMP    CASE, 0005          ; case 5 -> display the crlf
        JUMPZ   PRINT_CRLF

        COMP    CASE, 0006          ; case 6 -> display the count
        JUMPZ   PRINT_COUNT

        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_BS SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRINT_BS

        COMP    POINTER, end_bs
        RETURNC
        LOAD    CASE, 0000
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_BANNER SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRINT_BANNER

        COMP        POINTER, end_banner
        RETURNC
        LOAD        CASE, 0002      ; print prompt after
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_PROMPT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRINT_PROMPT
        COMP        POINTER, end_prompt
        RETURNC
        LOAD        CASE, 0000
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_HOMETOWN SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRINT_HOMETOWN

        COMP        POINTER, end_hometown
        RETURNC
        LOAD        POINTER, start_prompt
        LOAD        CASE, 0002
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_CRLF SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
PRINT_CRLF
        COMP        POINTER, end_crlf
        RETURNC
        LOAD        POINTER, start_prompt
        LOAD        CASE, 0002  ; print prompt after
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PRINT_COUNT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
PRINT_COUNT
        COMP    POINTER, COUNT_END
        RETURNC
        LOAD    POINTER, start_crlf
        LOAD    CASE, 0005
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GOT_RXRDY SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GOT_RXRDY
        COMP        CASE, 0000     ; z if its true
        RETURNNZ                   ; return if no z
        ; i.e. only enter this function if the user is inputted
        ; chars
        ; based on those chars, set the case and the pointer

        INPUT       DATA, 0000
        COMP        DATA, 0000 ; comp to null
        RETURNZ

        COMP        DATA, ASCII_aster
        JUMPZ       SET_HOMETOWN

        COMP        DATA, ASCII_BS
        JUMPZ       SET_BS

        COMP        DATA, ASCII_CR
        JUMPZ       SET_CRLF

        COMP        DATA, ASCII_AT
        JUMPZ       SET_AT

        ; This last condition is used for echoing
        ADD         CHAR_COUNT, 0001
        OUTPUT      DATA, 0000
        COMP        CHAR_COUNT, forty
        JUMPZ       SET_CRLF
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SET_HOMETOWN SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SET_HOMETOWN
        LOAD        CASE, 0003
        LOAD        POINTER, start_hometown
        LOAD        TEMP, ASCII_null
```

```
        OUTPUT      TEMP, 0000
        LOAD        CHAR_COUNT, 0000
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SET_BS SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SET_BS
        COMP        CHAR_COUNT, 0000
        RETURNZ
        LOAD        CASE, 0004
        LOAD        POINTER, start_bs
        LOAD        TEMP, ASCII_null
        OUTPUT      TEMP, 0000
        SUB         CHAR_COUNT, 0001
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SET_CRLF SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SET_CRLF

        LOAD        CASE, 0005
        LOAD        POINTER, start_crlf
        LOAD        TEMP, ASCII_null
        OUTPUT      TEMP, 0000
        LOAD        CHAR_COUNT, 0000
        RETURN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;SET_AT SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SET_AT
        CALL        BINARY_TO_ASCII
        LOAD        CASE, 0006
        LOAD        POINTER, COUNT_BEGIN
        LOAD        TEMP, ASCII_null
        OUTPUT      TEMP, 0000
        LOAD        CHAR_COUNT, 0000
        RETURN


        ADDRESS 0FFE
        JUMP    ISR
        END
```

33.2 Nexys-A7-100T-Receiver_Engine.xdc

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level
signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];


##Switches
#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { hex_4[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { OHEL }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { PEN }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { EIGHT }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { baud_value[0]
}]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { baud_value[1]
}]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { baud_value[2]
}]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { baud_value[3]
}]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports {
digit_select[0] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports {
digit_select[1] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports {
digit_select[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports {
digit_select[3] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports {
digit_select[4] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {
digit_select[5] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports {
digit_select[6] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports {
digit_select[7] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LEDs[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LEDs[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LEDs[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
```

```
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LEDs[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LEDs[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LEDs[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LEDs[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LEDs[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LEDs[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LEDs[9] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LEDs[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LEDs[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LEDs[12] }];
#IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LEDs[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LEDs[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LEDs[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Buttons
#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { hex4[0] }];
#IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { hex4[2] }];
#IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { hex4[3] }];
#IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { hex5[0] }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd

##USB-RS232 Interface
set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 } [get_ports { RX }];
#IO_L7P_T1_AD6P_35 Sch=uart_txd_in
set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 } [get_ports { TX }];
#IO_L11N_T1_SRCC_35 Sch=uart_rxd_out
#set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33 } [get_ports { UART_CTS }];
#IO_L12N_T1_MRCC_35 Sch=uart_cts
#set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33 } [get_ports { UART_RTS }];
#IO_L5N_T0_AD13N_35 Sch=uart_rts
```

33.3 UART_TSI_TOP.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: UART_TSI_TOP.v                                                    //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 12/08/2020 01:57:41 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module UART_TSI_TOP(
    input clk,              // 100MHz clock
    input reset,            // Synchronous reset
    input [3:0] baud_value, // Baud value select
    input RX,               // 1 bit receieved data
    input EIGHT,            // Eight bits input
    input PEN,              // Parity enable input
    input OHEL,             // Even/Odd parity select input
    output TX,              // 1 bit transmitted data
    output [15:0] LEDs      // On board LEDs
    );

    wire clk_wire;              // 100MHz clock
    wire reset_wire;            // Synchronous reset
    wire [3:0] baud_value_wire; // Baud value select
    wire RX_wire;               // 1 bit receieved data
    wire EIGHT_wire;            // Eight bits input
    wire PEN_wire;              // Parity enable input
    wire OHEL_wire;             // Even/Odd parity select input
    wire TX_wire;               // 1 bit transmitted data
    wire [15:0] LEDs_wire;      // On board LEDs

    TSI tsi_instance( .clk_I(clk),
                      .reset_I(reset),
                      .baud_value_I(baud_value),
                      .EIGHT_I(EIGHT),
                      .PEN_I(PEN),
                      .OHEL_I(OHEL),
                      .RX_I(RX),
                      .TX_I(TX_wire),
                      .LEDs_I(LEDs_wire),

                      .clk_O(clk_wire),
                      .reset_O(reset_wire),
                      .baud_value_O(baud_value_wire),
                      .EIGHT_O(EIGHT_wire),
                      .PEN_O(PEN_wire),
                      .OHEL_O(OHEL_wire),
                      .RX_O(RX_wire),
```

```
                        .TX_O(TX),
                        .LEDs_O(LEDs)
                        );

    UART uart_instance( .clk(clk_wire),
                        .reset(reset_wire),
                        .baud_value(baud_value_wire),
                        .EIGHT(EIGHT_wire),
                        .PEN(PEN_wire),
                        .OHEL(OHEL_wire),
                        .RX(RX_wire),
                        .TX(TX_wire),
                        .LEDs(LEDs_wire)
                        );
endmodule
```

33.4 TSI.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: TSI.v                                                             //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 12/08/2020 01:42:08 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module TSI(
    input       clk_I,
    input       reset_I,
    input [3:0] baud_value_I,
    input       EIGHT_I, PEN_I, OHEL_I,
    input       RX_I,
    input       TX_I,
    input [15:0] LEDs_I,

    output       clk_O,
    output       reset_O,
    output [3:0] baud_value_O,
    output       EIGHT_O, PEN_O, OHEL_O,
    output       RX_O,
    output       TX_O,
    output [15:0] LEDs_O
    );

    BUFG clk( .I(clk_I),
              .O(clk_O)
             );

    IBUF reset( .I(reset_I),
                .O(reset_O)
               );

    IBUF baud_value[3:0]( .I(baud_value_I),
                          .O(baud_value_O)
                         );

    IBUF EIGHT( .I(EIGHT_I),
                .O(EIGHT_O)
                );

    IBUF PEN( .I(PEN_I),
              .O(PEN_O)
             );

    IBUF OHEL( .I(OHEL_I),
```

```
                .O(OHEL_O)
                );

    IBUF RX( .I(RX_I),
             .O(RX_O)
            );

    OBUF TX( .I(TX_I),
             .O(TX_O)
            );

    OBUF LEDs[15:0]( .I(LEDs_I),
                     .O(LEDs_O)
                    );

endmodule
```

33.5 UART.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: UART.v                                                            //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 11/12/2020 06:52:36 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module UART(
    input clk,              // 100MHz clock
    input reset,            // Asynchronous reset
    input [3:0] baud_value, // Baud value select
    input RX,               // 1 bit receieved data
    input EIGHT,            // Eight bits input
    input PEN,              // Parity enable input
    input OHEL,             // Even/Odd parity select input
    output TX,              // 1 bit transmitted data
    output reg [15:0] LEDs  // On board LEDs
    );
    wire resetNew;          // Asynchronous reset with synchronous de-assertion
    wire [18:0] K;          // Total baud value
    wire [7:0] UART_RDATA;  // UART received data
    wire RXRDY;             // RX ready flag
    wire FERR;              // Framing error flag
    wire OVF;               // Overflow flag
    wire [15:0] OUT_PORT_wire; // TramelBlaze OUT_PORT wire
    wire TXRDY;             // TX ready flag
    wire [15:0] PORT_ID_wire; // TramelBlaze PORT_ID wire
    wire READ_STROBE_wire;  // TramelBlaze READ_STROBE wire
    wire WRITE_STROBE_wire; // TramelBlaze WRITE_STROBE wire
    wire [15:0] IN_PORT_wire; // TramelBlaze IN_PORT wire
    wire INTERRUPT_wire;    // TramelBlaze INTERRUPT wire
    wire INTERRUPT_ACK_wire; // TramelBlaze INTERRUPT_ACK wire
    wire ped_top_out;       // Top PED output for the RXRDY
    wire ped_bot_out;       // Bottom PED output for the TXRDY
    wire UART_INT;          // UART interrupt

    // Combo logic for the UART_INT
    assign UART_INT = ped_top_out | ped_bot_out;

    // Combo logic for the UART_DS
    assign IN_PORT_wire = (PORT_ID_wire == 16'h0001) ? {3'b000, OVF, FERR, PERR, TXRDY,
RXRDY} : UART_RDATA;

    // Instantiation for the SR Flip Flop
    // Inputs: clk, reset, S, R
    // Outputs: Q
```

```
SR_FF sr_uart( .clk(clk),
               .reset(resetNew),
               .S(UART_INT),
               .R(INTERRUPT_ACK_wire),
               .Q(INTERRUPT_wire)
              );

// Instantiation for the AISO
// Inputs: clk, reset
// Outputs: resetNew
AISO aiso( .clk(clk),
           .reset(reset),
           .resetNew(resetNew)
          );

// Instantiation for the BAUD DECODE
// Inputs: baud_value
// Outputs: baud
BAUD_DECODE bd( .baud_value(baud_value),
                .baud(K)
               );

// Instantiation for the RX Engine
// Inputs: clk, reset, RX, EIGHT, PEN, OHEL, K, READS0
// Outputs: UART_RDATA, RXRDY, PERR, FERR, OVF
RX_Engine rx( .clk(clk),
              .reset(resetNew),
              .RX(RX),
              .EIGHT(EIGHT),
              .PEN(EIGHT),
              .OHEL(OHEL),
              .K(K),
              .READS0( (PORT_ID_wire == 16'h0000) && READ_STROBE_wire),
              .UART_RDATA(UART_RDATA),
              .RXRDY(RXRDY),
              .PERR(PERR),
              .FERR(FERR),
              .OVF(OVF)
             );

// Instantiation for the TX Engine
// Inputs: clk, reset, write0, out_port, baud, EIGHT, PEN, OHEL
// Outputs: TXRDY, TX
transmitEngine tx( .clk(clk),
                   .reset(resetNew),
                   .write0( (PORT_ID_wire == 16'h0000) && WRITE_STROBE_wire),
                   .out_port( {8'b0,OUT_PORT_wire[7:0]}),
                   .baud(K),
                   .EIGHT(EIGHT),
                   .PEN(PEN),
                   .OHEL(OHEL),
                   .TXRDY(TXRDY),
                   .TX(TX)
                  );

// Instantiation for the TramelBlaze
// Inputs: CLK, RESET, IN_PORT, INTERRUPT
```

```
    // Outputs: OUT_PORT, PORT_ID, READ_STROBE, WRITE_STROBE, INTERRUPT_ACK
    tramelblaze_top tbt( .CLK(clk),
                         .RESET(resetNew),
                         .IN_PORT(IN_PORT_wire),
                         .INTERRUPT(INTERRUPT_wire),
                         .OUT_PORT(OUT_PORT_wire),
                         .PORT_ID(PORT_ID_wire),
                         .READ_STROBE(READ_STROBE_wire),
                         .WRITE_STROBE(WRITE_STROBE_wire),
                         .INTERRUPT_ACK(INTERRUPT_ACK_wire)
                        );

    // Instantiation for the PED for the RX engine
    // Inputs: clk, reset, in
    // Outputs: ped
    ped2 ptop( .clk(clk),
               .reset(resetNew),
               .in(RXRDY),
               .ped(ped_top_out)
              );

    // Instantiation for the PED for the TX engine
    // Inputs: clk, reset, in
    // Outputs: ped
    ped2 pbot( .clk(clk),
               .reset(resetNew),
               .in(TXRDY),
               .ped(ped_bot_out)
              );

    // Sequential logic for the LEDs
    always@(posedge clk, posedge resetNew)
        if (resetNew)
            LEDs <= 16'b0;
        else if (PORT_ID_wire == 16'h0001 && WRITE_STROBE_wire)
            LEDs <= OUT_PORT_wire;
endmodule
```

33.6 AISO.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: AISO.v                                                            //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 02/18/2020 12:15:46 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////
module AISO(
    input  clk,      //input clk
    input  reset,    //input asynchronous reset
    output resetNew //Output for the for synchornous reset
    );
    reg Q_Meta; //input flop (D)
    reg Q;      //output flop (Q)

    assign resetNew = ~Q; //combo logic to negate Q

    //sequential logic
    always@(posedge clk, posedge reset)
        if (reset)
            Q_Meta <= 1'b0;
        else
            Q_Meta <= 1'b1;

    //HANEY WAY
    always@(posedge clk)
            Q <= Q_Meta;

endmodule
```

33.7 ped2.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: ped2.v                                                            //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 12/18/2020 12:30:05 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module ped2(
    input clk,
    input reset,
    input in,  //input to check for postive edge detect
    output ped //output postitve edge detected
    );

    reg q1 , q2;  //output flops (D)
    reg nq1, nq2; //input flops (Q)

    //combo logic
    assign ped = q1 & ~q2;

    always@(*)
        begin
        nq1 = in;
        nq2 = q1;
    end

    //Sequential Logic
    always@(posedge clk, posedge reset)
        if(reset) {q1,q2} <= 2'b0;
        else      {q1,q2} <= {nq1,nq2};

endmodule
```

33.8 SR_FF.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: SR_FF.v                                                           //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 09/23/2020 01:01:30 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module SR_FF(
    input clk,      // 100MHz clock
    input reset,    // Asynchronous reset with synchronous de-assertion
    input S,        // SET input for the SR flop
    input R,        // RESET input for the SR flop
    output reg Q    // Output for SR flip flop
    );

    // Sequential logic
    always@(posedge clk, posedge reset)
        if (reset)      // If reset
            Q <= 1'b0;  // Q <= 0
        else if (S)     // else if S
            Q <= 1'b1;  // Q <= 1
        else if (R)     // else if R
            Q <= 1'b0;  // R <= 0
        else            // else
            Q <= Q;     // Q <= Q

endmodule
```

33.9 BAUD_DECODE.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: BAUD_DECODE.v                                                     //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 11/12/2020 07:11:02 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module BAUD_DECODE(
    input [3:0] baud_value,
    output reg [18:0] baud
    );

    always@(*)
        case(baud_value)             // BAUD RATE VALUES
            4'b0000: baud = 333333; // 300
            4'b0001: baud = 83333;  // 1200
            4'b0010: baud = 41667;  // 2400
            4'b0011: baud = 20833;  // 4800
            4'b0100: baud = 10417;  // 9600
            4'b0101: baud = 5208;   // 19200
            4'b0110: baud = 2604;   // 38400
            4'b0111: baud = 1736;   // 57600
            4'b1000: baud = 868;    // 115200
            4'b1001: baud = 434;    // 230400
            4'b1010: baud = 217;    // 460800
            4'b1011: baud = 109;    // 92160
            default: baud = 0;      // 0
        endcase
endmodule
```

33.10 transmitEngine.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// File name: transmitEngine.v                                  //
//                                                              //
// Created by       Jesus Franco                                //
// Copyright © 2020 Jesus Franco on 10/15/2020 03:08:28 PM      //
//                                                              //
//                                                              //
// In submitting this file for class work at CSULB              //
// I am confirming that this is my work and the work            //
// of no one else. In submitting this code I acknowledge that   //
// plagiarism in student project work is subject to dismissal.  //
// from the class                                               //
// Dependencies:                                                //
//////////////////////////////////////////////////////////////////

module transmitEngine(
    input clk,              // 100MHz clock
    input reset,            // asynchronous reset with synchronous de-assertion
    input write0,           // load from address decoders
    input [7:0] out_port,   // out_port wire to output data from .coe file
    input [18:0] baud,
    input EIGHT,            // eigth enabled bit
    input PEN,              // parity enable
    input OHEL,             // even/odd parity enable
    output TXRDY,           // TXRDY output from the SSR flop
    output TX               // TX output from the shift register
    );
    wire done;              // done wire from bit counter
    wire doit;              // doit wire for the muxes select
    wire dff_out;           // output wire from the DFF
    wire [7:0] load_out;    // output wire from the LoadReg_8bit
    wire btu;               // btu wire for the muxes select
    wire [1:0] decode_out;  // output wire from the decode

    // Instantiation for the SSR Flip Flop
    // Inputs: clk, reset, S, R
    // Outputs: Q
    SSR_FF ssr( .clk(clk),
                .reset(reset),
                .S(done),
                .R(write0),
                .Q(TXRDY)
                );

    // Instantiation for the SR Flip Flop
    // Inputs: clk, reset, S, R
    // Outputs: Q
    SR_FF sr( .clk(clk),
              .reset(reset),
              .S(write0),
              .R(done),
              .Q(doit)
              );
    // Instantiation for the D Flip Flop
```

```
// Inputs: clk, reset, D
// Outputs: Q
DFF df( .clk(clk),
        .reset(reset),
        .D(write0),
        .Q(dff_out)
        );

// Instantiation for the bit counter with the baud select mux
// Inputs: clk, reset, doit, baud_value
// Outputs: btu
bitCounter_baud bcb( .clk(clk),
                     .reset(reset),
                     .doit(doit),
                     .baud(baud),
                     .btu(btu)
                     );

// Instantiation for the bit counter
// Inputs: clk, reset, doit, btu
// Outputs: done
bitCounter bc( .clk(clk),
               .reset(reset),
               .doit(doit),
               .btu(btu),
               .done(done)
               );

// Instantiation for the 8 bit loadable register
// Inputs: clk, reset, load, D
// Outputs: Q
LoadReg_8bit ld8( .clk(clk),
                  .reset(reset),
                  .load(write0),
                  .D(out_port),
                  .Q(load_out)
                  );

// Instantiation for the decoder
// Inputs: clk, reset, eight, pen, ohel, data
// Outputs: dout
decode dc( .eight(EIGHT),
           .pen(PEN),
           .ohel(OHEL),
           .data(load_out),
           .dout(decode_out)
           );
// Instantiation for the shift register
// Inputs: clk, reset, ld, sh, bit9, bit10, din
// Outputs: tx
shiftreg shr( .clk(clk),
              .reset(reset),
              .ld(dff_out),
              .sh(btu),
              .bit9(decode_out[0]),
              .bit10(decode_out[1]),
              .din(load_out[6:0]),
```

```
                .tx(TX)
                );

endmodule
```

33.11 SSR_FF.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: SSR_FF.v                                                          //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/17/2020 04:06:22 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module SSR_FF(
    input clk,      // 100MHz clock
    input reset,    // Asynchronous reset with synchronous de-assertion
    input S,        // SET input for the SR flop
    input R,        // RESET input for the SR flop
    output reg Q    // Output for SR flip flop
    );

// Sequential logic
always@(posedge clk, posedge reset)
    if (reset)      // If reset
        Q <= 1'b1;  // Q <= 1
    else if (S)     // else if S
        Q <= 1'b1;  // Q <= 1
    else if (R)     // else if R
        Q <= 1'b0;  // R <= 0
    else            // else
        Q <= Q;     // Q <= Q
Endmodule
```

33.12 DFF.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: DFF.v                                                             //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/13/2020 03:46:36 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module DFF(
    input clk,   // 100MHz clock
    input reset, // Asynchronous reset with synchronous de-assertion
    input D,     // holding value
    output reg Q // output value
    );

    // Sequential logic
    always@(posedge clk, posedge reset)
        if(reset)       // if reset
            Q <= 1'b0; // Q <= 0
        else
            Q <= D;     // Q <= D

endmodule
```

33.13 bitCounter_baud.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: bitCounter_baud.v                                                 //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/15/2020 01:44:00 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module bitCounter_baud(
    input clk,              // 100MHz clock
    input reset,            // asynchronous reset with synchronous de-assertion
    input doit,             // 1/2 of the select for the mux
    input [18:0] baud,
    //input [3:0] baud_value, // baud value select
    output wire btu         // 1/2 of the select for the mux
    );
reg [18:0] Q, D;            // I/O flops
//reg [18:0] baud;            // mux out register named baud

// btu logic will output '1' if counter reaches baud total value, else '0'
assign btu = (Q == baud) ? 1'b1 : 1'b0;

// Sequential logic of a 19 bit flip flop
always@(posedge clk, posedge reset)
    if(reset)
        Q <= 19'b0;
    else
        Q <= D;

// Combo logic for the bit counter 2to1 mux
always@(*)
    case ( {doit, btu} )
        2'b00  : D = 19'b0;
        2'b01  : D = 19'b0;
        2'b10  : D = Q + 19'b1;
        2'b11  : D = 19'b0;
        default: D = 19'b0;
    endcase

endmodule
```

33.14 bitCounter.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: bitCounter.v                                                      //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/15/2020 01:30:52 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module bitCounter(
input clk,  // 100MHz clock
input reset,// asynchronous reset with synchronous de-assertion
input doit, // 1/2 of the select for the mux
input btu,  // 1/2 of the select for the mux
output done // output done if counter reaches 4'd11
);
reg [3:0] D, Q; // 4 bit I/O flops

// done logic will output '1' if counter reaches 4'd11, else '0'
assign done = (Q == 4'b1011)? 1'b1: 1'b0;

// Sequential logic
always@(posedge clk, posedge reset)
    if(reset)
        Q <= 4'b0;
    else
        Q <= D;

// Combo logic for 2to1 mux
always@(*)
    case ( {doit, btu} )
        2'b00  : D = 1'b0;
        2'b01  : D = 1'b0;
        2'b10  : D = Q;
        2'b11  : D = Q + 4'b1;
        default: D = 4'b0;
    endcase

endmodule
```

33.15 LoadReg_8bit.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: LoadReg_8bit.v                                                    //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/13/2020 03:44:22 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module LoadReg_8bit(
    input clk,          // 100MHz clock
    input reset,        // Asynchronous reset with synchronous de-assertion
    input load,         // load enable for flop
    input [7:0] D,     // holding value
    output reg [7:0] Q // output value
    );

// Sequential logic
always@(posedge clk, posedge reset)
    if(reset)       // if reset
        Q <= 8'b0; // Q <= 0
    else if(load)   // else if load
        Q <= D;     // Q <= D
    else            // else
        Q <= Q;     // Q <= Q

endmodule
```

33.16 decode.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: decode.v                                                          //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/13/2020 02:46:50 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module decode(
    input eight,            // eighth bit enable
    input pen,              // parity enable
    input ohel,             // even/odd parity enable
    input [7:0] data,       // recieving data
    output reg [1:0] dout   // data out
    );

    // Combo logic for the decoder
    always@(*)
        case({eight,pen,ohel})
            3'b000: dout = 2'b11;                   // [ 1, 1 ]
            3'b001: dout = 2'b11;                   // [ 1, 1 ]
            3'b010: dout = {1'b1, ^data[6:0]};      // [ 1, even parity 6-0 ]
            3'b011: dout = {1'b1, ~^data[6:0]};     // [ 1, odd  parity 6-0 ]
            3'b100: dout = {1'b1, data[7]};         // [ 1, data7 ]
            3'b101: dout = {1'b1, data[7]};         // [ 1, data7 ]
            3'b110: dout = {^data[7:0], data[7]};   // [ even parity 7-0, data7 ]
            3'b111: dout = {~^data[7:0], data[7]};  // [ odd parity 7-0 , data7 ]
            default:dout = 2'b00;                   // [ 0, 0 ]
        endcase

endmodule
```

33.17 shiftreg.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: shiftreg.v                                                        //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/13/2020 02:26:05 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module shiftreg(
    input       clk,    // 100MHz clock
    input       reset,  // asynchronous reset with synchronous de-assertion
    input       ld,     // load enable
    input       sh,     // shift enable
    input       bit9,   // bit 9
    input       bit10,  // bit 10
    input [6:0] din,    // data in
    output wire tx      // output transmitted 1 bit data
    );
    reg   [10:0] Shifter; // shifting register

    ///////////////
    // define transmit out
    ///////////////
    assign tx = Shifter[0]; // tx will be output based on Shifter[0]

    ///////////////
    // define shift register
    ///////////////
    // Sequential logic for the shift register
    always@(posedge clk, posedge reset)
        if(reset) Shifter <= 11'h7FF;                          else
        if(ld)    Shifter <= {bit10, bit9, din[6:0], 2'b01}; else
        if(sh)    Shifter <= {1'b1, Shifter[10:1]};          else
                  Shifter <= Shifter;
endmodule
```

33.18 RX_Engine.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: RX_Engine.v                                                       //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 11/09/2020 01:50:55 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module RX_Engine(
    input clk,      // 100MHz clock
    input reset,    // Asynchronous reset with synchronous de-assertion
    input RX,       // 1 bit recieved data
    input EIGHT,    // Eight bit enable
    input PEN,      // Parity enable
    input OHEL,     // Even/Odd enable
    input [18:0] K, // Baud value
    input READS0,   // Clear
    output [7:0] UART_RDATA, // 8 bit UART received data
    output RXRDY,   // RX ready flag
    output PERR,    // Parity error flag
    output FERR,    // Framing error flag
    output OVF      // Overflow error flag
    );

    reg [1:0] ps, ns;   // present state and next state
    reg START;      // START bit
    reg DOIT;       // DOIT for the DOIT counter

////////////////////////////////////////////////// FSM
    always@(posedge clk, posedge reset)
        if(reset) ps <= 2'b0;
        else      ps <= ns;

    always@(*)
        case(ps)
            2'b00:
                begin
                    START = 0;
                    DOIT = 0;
                    if(RX)
                        ns <= 2'b00;
                    else if(~RX)
                        ns <= 2'b01;
                end
            2'b01:
                begin
                    START = 1;
```

```
                    DOIT = 1;
                    if(~RX & ~BTU)
                        ns <= 2'b01;
                    else if(~RX & BTU)
                        ns <= 2'b10;
                    else if(RX)
                        ns <= 2'b00;
                end
            2'b10:
                begin
                    START = 0;
                    DOIT = 1;
                    if(~DONE)
                        ns <= 2'b10;
                    else if(DONE)
                        ns <= 2'b00;
                end

            default:
                begin
                    ns <= 2'b00;
                    START = 0;
                    DOIT = 0;
                end
        endcase

////////////////////////////// bit counter for DONE
    reg [3:0] bitcount_Q, bitcount_D;
    reg [3:0] D;

    assign DONE = (bitcount_Q == D);

    always@(posedge clk, posedge reset)
        if(reset) bitcount_Q <= 4'b0;
        else      bitcount_Q <= bitcount_D;

    always@(*)
        case({DOIT,BTU})
            2'b00: bitcount_D <= 4'b0;
            2'b01: bitcount_D <= 4'b0;
            2'b10: bitcount_D <= bitcount_Q;
            2'b11: bitcount_D <= bitcount_Q + 4'b1;
            default: bitcount_D <= 4'b0;
        endcase

    always@(*)
        case({EIGHT,PEN})
            2'b00: D <= 9;
            2'b01: D <= 10;
            2'b10: D <= 10;
            2'b11: D <= 11;
            default: D <= 9;
        endcase

////////////////////////////// bit time counter for BTU
    reg [18:0] bittimecount_D, bittimecount_Q;
    reg [18:0] bittimecount;
```

```
    assign BTU = (bittimecount_Q == bittimecount);

    always@(posedge clk, posedge reset)
        if(reset) bittimecount_Q <= 19'b0;
        else      bittimecount_Q <= bittimecount_D;

    always@(*)
        case({DOIT,BTU})
            2'b00: bittimecount_D <= 19'b0;
            2'b01: bittimecount_D <= 19'b0;
            2'b10: bittimecount_D <= bittimecount_Q + 19'b1;
            2'b11: bittimecount_D <= 19'b0;
            default: bittimecount_D <= 19'b0;
        endcase

    always@(*)
        if(START) bittimecount <= K >> 1;
        else      bittimecount <= K;

////////////////////////////// SHIFT REGISTER
    wire SHIFT;
    //reg SDI; // not used
    reg [9:0] Shifter;

    assign SHIFT = BTU & ~START;

    always@(posedge clk, posedge reset)
        if(reset) Shifter <= 10'b0;
        else if(SHIFT) Shifter <= {RX, Shifter[9:1]};

////////////////////////////// RIGHT JUSTIFY
    reg [9:0] right_justify;

    // Will account for 8 or 7 bits outputs
    assign UART_RDATA = (EIGHT)  ? right_justify[7:0] : {1'b0, right_justify[6:0]};
    // Will account for 8 bits outputs
    //assign UART_RDATA = right_justify[7:0];

    always@(*)
        case({EIGHT,PEN})
            2'b00: right_justify <= {2'b11, Shifter[9:2]};
            2'b01: right_justify <= {1'b1,  Shifter[9:1]};
            2'b10: right_justify <= {1'b1,  Shifter[9:1]};
            2'b11: right_justify <= Shifter;
            default: right_justify <= 10'b0;
        endcase

////////////////////////////// GENERATED PARITY SELECT
    reg top_mux_out;
    reg mid_mux_out;
    reg bot_mux_out;

    reg xor1_mux_out;

    wire XOR_1;
    wire XOR_2;
```

```
    // TOP MUX for generated parity select
    always@(*)
        if(EIGHT) top_mux_out <= right_justify[7];
        else      top_mux_out <= 1'b0;

    assign XOR_1 = right_justify[6:0] ^ top_mux_out;

    always@(*)
        if(OHEL)
            xor1_mux_out <= ~XOR_1;
        else
            xor1_mux_out <=  XOR_1;

    assign XOR_2 = xor1_mux_out ^ mid_mux_out;

    // MID MUX for received parity select
    always@(*)
        if(EIGHT) mid_mux_out <= right_justify[8];
        else      mid_mux_out <= right_justify[7];
    // BOT MUX stop bit select
    always@(*)
        case({EIGHT,PEN})
            2'b00: bot_mux_out <= right_justify[7];
            2'b01: bot_mux_out <= right_justify[8];
            2'b10: bot_mux_out <= right_justify[8];
            2'b11: bot_mux_out <= right_justify[9];
            default: bot_mux_out <= 1'b0;
        endcase

////////////////////////////// SR FFs
    //wire RXRDY;
    // SR for RXRDY
    SR_FF sr0( .clk(clk),
               .reset(reset),
               .S(DONE),
               .R(READS0),
               .Q(RXRDY)
               );
    //wire PERR;
    // SR for PERR
    SR_FF sr1( .clk(clk),
               .reset(reset),
               .S(PEN & XOR_2 & DONE),
               .R(READS0),
               .Q(PERR)
               );
    //wire FERR;
    // SR for FERR
    SR_FF sr2( .clk(clk),
               .reset(reset),
               .S(DONE & ~bot_mux_out),
               .R(READS0),
               .Q(FERR)
               );
    //wire OVF;
    // SR for FERR
```

```
    SR_FF sr3( .clk(clk),
               .reset(reset),
               .S(DONE & RXRDY),
               .R(READS0),
               .Q(OVF)
               );

endmodule
```

33.19 transmitEngine_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// File name: transmitEngine_tb.v                               //
//                                                              //
// Created by       Jesus Franco                                //
// Copyright © 2020 Jesus Franco on 10/17/2020 01:50:05 PM      //
//                                                              //
//                                                              //
// In submitting this file for class work at CSULB              //
// I am confirming that this is my work and the work            //
// of no one else. In submitting this code I acknowledge that   //
// plagiarism in student project work is subject to dismissal.  //
// from the class                                               //
// Dependencies:                                                //
//////////////////////////////////////////////////////////////////

module transmitEngine_tb;
    reg clk;
    reg reset;
    reg write0;
    reg [7:0] out_port;
    reg [18:0] baud;
    reg EIGHT;
    reg PEN;
    reg OHEL;
    wire TXRDY;
    wire TX;

    transmitEngine uut( .clk(clk),
                        .reset(reset),
                        .write0(write0),
                        .out_port(out_port),
                        .baud(baud),
                        .EIGHT(EIGHT),
                        .PEN(PEN),
                        .OHEL(OHEL),
                        .TXRDY(TXRDY),
                        .TX(TX)
                        );

    always #5 clk = ~clk;
    initial begin
        clk = 0;
        reset = 1;
        baud = 19'd109;
        EIGHT = 1;
        PEN = 1;
        OHEL = 1;
        out_port = 8'b00000111;
        write0 = 0;

        #100;
        reset = 0;

        #100;
```

```
        write0 = 1;
        #10;
        write0 = 0;

    end

endmodule
```

33.20 TramelBlazePlusTransmit_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: TramelBlazePlusTransmit_tb.v                                      //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 10/17/2020 01:26:55 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module TramelBlazePlusTransmit_tb;
    reg clk;
    reg reset;
    reg [3:0] baud_value;
    reg EIGHT;
    reg PEN;
    reg OHEL;
    wire TX;
    wire [15:0] LEDs;

    TramelBlazePlusTransmit tbpt( .clk(clk),
                                  .reset(reset),
                                  .baud_value(baud_value),
                                  .EIGHT(EIGHT),
                                  .PEN(PEN),
                                  .OHEL(OHEL),
                                  .TX(TX),
                                  .LEDs(LEDs)
                                  );

    always #5 clk = ~clk;
    initial begin
        clk = 0;
        reset = 1;
        baud_value = 4'b1011;
        EIGHT = 1;
        PEN = 0;
        OHEL = 0;

        #100;
        reset = 0;
    end

endmodule
```

33.21 RX_Engine_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: RX_Engine_tb.v                                                    //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 11/10/2020 08:08:31 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module RX_Engine_tb;
	reg clk;
	reg reset;
	reg READS0;
	reg RX;
	reg EIGHT;
	reg PEN;
	reg OHEL;
	reg [18:0] K;

	wire [7:0] UART_RDATA;
	wire RXRDY;
	wire PERR;
	wire FERR;
	wire OVF;

	RX_Engine uut (
		.clk(clk),
		.reset(reset),
		.READS0(READS0),
		.RX(RX),
		.EIGHT(EIGHT),
		.PEN(PEN),
		.OHEL(OHEL),
		.K(K),
		.UART_RDATA(UART_RDATA),
		.RXRDY(RXRDY),
		.PERR(PERR),
		.FERR(FERR),
		.OVF(OVF)
	);



   always #5 clk = ~clk;

	initial begin
		// Initialize Inputs
		clk = 0;
```

```
        reset = 1;
        READS0 = 0;
        RX = 1;
        EIGHT = 1;
        PEN = 0;
        OHEL = 0;
        K = 109 - 1;

        #100;
    reset = 0;
    RX = 0;

    // START
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b0;

    // 1
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b1;

    // 10
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b0;

    // 101
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b1;

    // 1010_
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b0;

    // 1010_1
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b1;

    // 1010_10
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b0;

    // 1010_101
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b1;

    // 1010_1010
    wait (uut.BTU == 1);
    wait (uut.BTU == 0);
    RX = 1'b0;
```

```
        // STOP bit
        wait (uut.BTU == 1);
        wait (uut.BTU == 0);
        RX = 1'b1;

        wait (uut.DONE == 1);
        #400;
        $stop;

        end
endmodule
```

33.22 UART_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// File name: UART_tb.v                                                         //
//                                                                              //
// Created by       Jesus Franco                                                //
// Copyright © 2020 Jesus Franco on 11/10/2020 08:08:31 PM                      //
//                                                                              //
//                                                                              //
// In submitting this file for class work at CSULB                              //
// I am confirming that this is my work and the work                            //
// of no one else. In submitting this code I acknowledge that                   //
// plagiarism in student project work is subject to dismissal.                  //
// from the class                                                               //
// Dependencies:                                                                //
//////////////////////////////////////////////////////////////////////////////////

module UART_tb;

	// Inputs
	reg clk;
	reg reset;
	reg [3:0] baud_value;
	reg EIGHT;
	reg PEN;
	reg OHEL;
	reg RX;

	// Outputs
	wire TX;
	wire [15:0] LEDs;

	// Instantiate the Unit Under Test (UUT)
	UART uut (
		.clk(clk),
		.reset(reset),
		.baud_value(baud_value),
		.EIGHT(EIGHT),
		.PEN(PEN),
		.OHEL(OHEL),
		.RX(RX),
		.TX(TX),
		.LEDs(LEDs)
	);



   always #5 clk = ~clk;

	initial begin
		// Initialize Inputs
		clk = 0;
		reset = 1;
	  baud_value = 4'b1011;
		RX = 1;
		EIGHT = 1;
```

```
            PEN = 0;
            OHEL = 0;


            // Wait 100 ns for global reset to finish
            #100;

      reset = 0;
      RX = 0; //start


      end

endmodule
```