2018

College of Engineering

Department of Computer Engineering and Computer Science

UART Full Duplex Protocol and Technology Specific Instantiation Chip Specification

PREPARED FOR
Mr. John Tramel
CECS 460 System-On-Chip Design

PREPARED BY
Jose Sotelo

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

Table of Contents

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

1. Introduction

The Universal Asynchronous Receiver Transmitter (UART) is a very common useful full duplex serial communication protocol. It serves as a basis for many ubiquitous protocols such as RS-232 (COM ports in computers). Its basis is a two-wire communication on which one port transmits and the other receives. The basic data units transferred is 1-byte. Data is transferred from the data bus to the transmitting UART in parallel from. Data that is been transmitted is organized into packets. Each packet contains 1 start bit, 5-9 data bits, an optional parity bit and 1 or 2 stop bits. The transferring speed of the data depends on the baud rate. The baud rate is a unit of measurement of bits per second (bps). Once the baud rate is selected, the data packet will be transferred to the receiving pin.

During the transmitting stage, the UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit. The data put together is output serially, bit by bit at the Tx pin. The receiving UART read the data bit by bit at its Rx pin. Once the data has been received, it's then converted back into parallel form and removes the start bit, parity but and stop bits. Lastly, the receiving UART transfers the data back in parallel to the data bus on the receiving end.

In Revision 2, we interface the full duplex UART circuit with the technology specific instantiation (TSI) circuit. The TSI contains all references to the target technology library. All the communications between the inputs and outputs of the device pass through the TSI. The I/O cells for a design are one of the primary inclusions of the TIS block. Each I/O of the chip must have a particularly selected device to meet electrical and timing requirements of the external interface. We will be using Nexys Artix-7 Library Guide which defines the cells and their capabilities.

1.1 Purpose

For this project, we will be designing an SOPC (System on Programmable Chip) that contains the Transmit Engine, Receive Engine and technology specific instantiation (TSI). The two engines created in this chip specification will make the full duplex UART protocol. The user will be able to configure the transfer of 8 or 7 bits, a parity bit, odd or even bit and the baud rate. We will interface the Transmit and Receive Engine with the 16-bit TramelBlaze and the TSI to send and receive data via a serial terminal RealTerm. The serial terminal will output a Banner, prompt the user, display the hometown of the designer, delete and input characters, and print the number of characters in the current line.

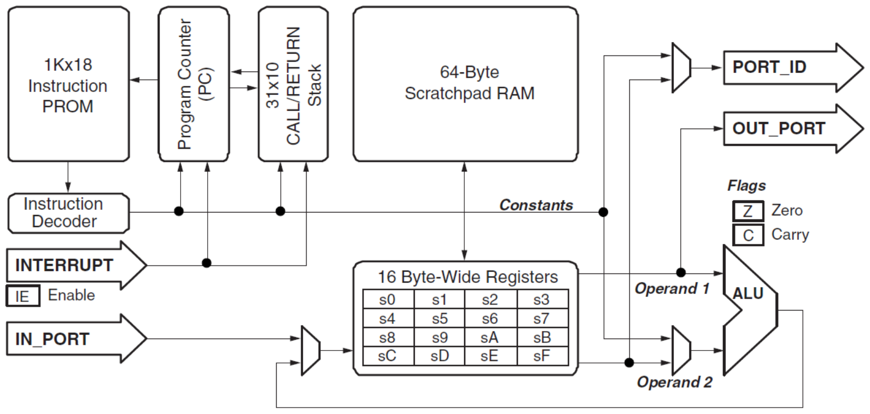
2. Applicable Documents:

2.1 External Documents

These documents were used in the development of the full duplex UART protocol. There main purpose was to assist the designer in the understanding of the embedded microcontroller the TramelBlaze and how to interface it with other FPGA logic.

2.2 PicoBlaze 8-bit Microcontroller User's Guide

The PicoBlaze^{TM} embedded microcontroller is an efficient, cost-effective embedded processor core for Spartan -3, Virtex -II, and Virtex-II Pro PFGAs. It is natively hosted on the Nexys 3/6 families. The 8-bit PicoBlaze microcontroller is specially designed and optimized for the Spartan -3, Virtex -II, and Virtex-II Pro architectures thus allowing for other FPGA logic to be connect to an embedded microcontroller's input and output ports.



2.3 TramelBlaze

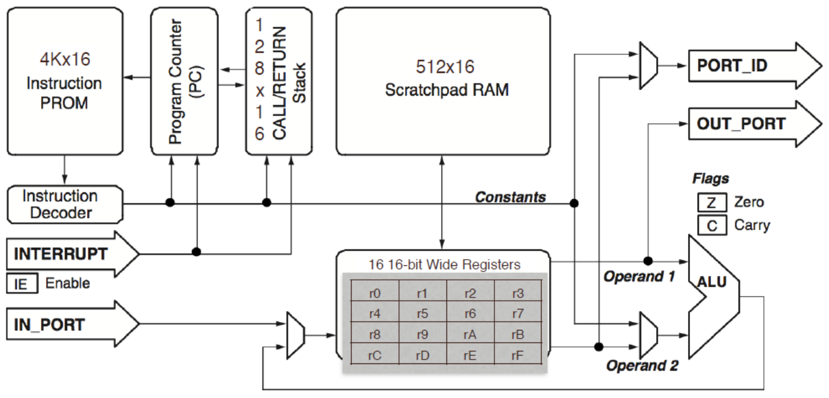
The TrambelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze. We will be using the TramelBlaze to interface it with develop FPGA logic. Since we switch on to the Nexys 4 FPGA, the PicoBlaze is no longer supported. Thus, the TramelBlaze was developed by John Tramel to allow the student to interface the embedded microcontroller with their own developed FPGA logic. The instruction set for the TramelBlaze is the PicoBlaze instruction set.

The TramelBlaze has the following:

- 16 16-bit wide general-purpose registers (r0-rF)
- 4096 Instruction Program Store - The code ROM provides storage for a suitable number of instructions. The instructions are either one or two words each.

Prepared by:	Date:	Revision:
Jose Sotelo	May 14, 2018	2.1

- Arithmetic Logic Unit (ALU) – 16-bit wide ALU performs all microcontroller operations such as basic arithmetic, bitwise logic operations, arithmetic compare, comprehensive shift and rotate operations, program control operation such as jump or call.
- Flags – The Zero flag indicates the result of the previous operation was zero and the CARRY flag indicates various conditions. The INTERRUPT_ENABLE flag indicates interrupt operations.
- 512-word scratchpad – A RAM within the processor with access provided by STORE and FETCH instructions. STORE copies the contents of a register to the RAM and FETCH copies the contents of the RAM to a register.
- I/O – the Input/Output ports extends the TramelBlaze's capabilities to allow interfacing to other PFGA logic. The addressing capabilities of the I/O instructions is 0 to 65535 (FFFF) which will require an external address decoder. INPUT operations move data from surrounding FPGA logic into a register and the OUTPUT operations move data from a register to the surrounding FPGA logic.
- Program Counter – The PC points to the next instruction to be executed. JUMP, CALL, RETURN, RETURNI instructions and interrupt or rest events will modify the contents of the PC.



Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

2.4 Programming TramelBlaze

	Instruction	Description	Function	ZERO	CARRY
0	**NOP**	Do Nothing		-	-
1	**ADD rX, kk**	Add register rX with literal kk	rX <– rX + kk	?	?
2	**ADD rX, rY**	Add register rX with register rY	rX <– rX + rY	?	?
3	**ADDCY rx, kk**	Add register rX with literal kk with CARRY	rX <– rX + kk + CARRY	?	?
4	**ADDCY rX, rY**	Add register rX with register rY with CARRY	rX <– rX + rY + CARRY	?	?
5	**AND rX, kk**	Bitwise AND register rX with literal kk	rX <– rX & kk	?	?
6	**AND rX, rY**	Bitwise AND register rX with register rY	rX <– rX & rY	?	?
7	**CALL aaa**	Unconditionally call subroutine at aaa	TOS <– PC; PC <– aaa	-	-
8	**CALLC aaa**	If CARRY call subroutine at aaa	if CARRY TOS <– PC; PC <– aaa	-	-
9	**CALLNC aaa**	If NOT CARRY call subroutine at aaa	if !CARRY TOS <– PC; PC <– aaa	-	-
10	**CALLZ aaa**	If ZERO call subroutine at aaa	if ZERO TOS <– PC; PC <– aaa	-	-
11	**CALLNZ aaa**	If NOT ZERO call subroutine at aaa	if !ZERO TOS <– PC; PC <– aaa	-	-
12	**COMP rX, kk**	Compare register rX with literal kk.	If rX== kk then ZERO =1; if rx <kk then CARRY = 1	?	?
13	**COMP rX, rY**	Compare register rX with register rY	If rX== rY then ZERO =1; if rx <rY then CARRY = 1	?	?
14	**DISINT**	Disable interrupt input	Interrupt Enable = 0	-	-
15	**ENINT**	Enable interrupt input	Interrupt Enable = 1	-	-
16	**INPUT rX, (rY)**	Read input port pointed to by rY into register rX	rX <– PORT(rY)	-	-
17	**INPUT rx, pp**	Read input port pp into register rX	rX <– PORT(pp)	-	-
18	**JUMP aaa**	Unconditionally jump to aaa	PC <– aaa	-	-
19	**JUMPC aaa**	Jump if CARRY to aaa	if CARRY PC <– aaa	-	-
20	**JUMPNC aaa**	Jump if NOT CARRY to aaa	if !CARRY PC <– aaa	-	-
21	**JUMPZ aaa**	Jump if ZERO to aaa	if ZERO PC <– aaa	-	-
22	**JUMPNZ aaa**	Jump if NOT ZERO to aaa	if ! ZERO PC <– aaa	-	-
23	**LOAD rX, kk**	Load register rX with literal kk	rX <– kk	-	-
24	**LOAD rX, rY**	Load register rX with contents of register rY	rX <– rY	-	-
25	**OR rX, kk**	Logically OR register rX with literal kk	rX <– rX | kk	?	0
26	**OR rX, rY**	Logically OR register rX with register rY	rX <– rX | rY	?	0
27	**OUTPUT rX, (rY)**	Write register rX to port pointed to by register rY	PORT(rY) <– rX	-	-
28	**OUTPUT rX, pp**	Write register rX to port pp	PORT(pp) <– rX	-	-
29	**RETURN**	Unconditionally return from subroutine	PC <– TOS+1	-	-
30	**RETURNC**	If CARRY return from subroutine	if CARRY PC <– TOS+1	-	-
31	**RETURNNC**	If NOT CARRY return from subroutine	if !CARRY PC <– TOS+1	-	-
32	**RETURNZ**	if ZERO return from subroutine	if ZERO PC <– TOS+1	-	-
33	**RETURNNZ**	if NOT ZERO return from subroutine	if !ZERO PC <– TOS+1	-	-
34	**RETDIS**	Return from interrupt with interrupts disabled	PC <– TOS+1; Interrupt Enable <– 0	?	?
35	**RETEN**	Return from interrupt with interrupts enabled	PC <– TOS+1; Interrupt Enable <– 1	?	?
36	**RL rX**	Rotate register rX left	rX <– {rX[14:0],rX[15]}; CARRY <– rX[15]	?	?
37	**RR rX**	Rotate register rX right	rX <– {rX[0],rX[15:1]}; CARRY <– rX[0]	?	?
38	**SL0 rX**	Shift register rX left, zero fill	rX <– {rX[14:0],0}; CARRY <– rX[15]	?	?
39	**SL1 rX**	Shift register rX left, one fill	rX <– {rX[14:0],1}; CARRY <– rX[15]	0	?
40	**SLA rX**	Shift register rX left through all bits, include CARRY	rX <– {rX[14:0], CARRY}; CARRY <– rX[15]	?	?
41	**SLX rX**	Shift register rX left. Bit rX[0] is unaffected	rX <– {rX[14:0],rX[0]}; CARRY <- rX[15]	?	?
42	**SR0 rX**	Shift register rX right, zero fill	rX <– {0,rX[15:1]}; CARRY <– rX[0]	?	?
43	**SR1 rX**	Shift register rX right, one fill	rX <– {1,rX[15:1]}; CARRY <– rX[0]	?	?
44	**SRA rX**	Shift register rX right through all bits, include CARRY	rX <– {CARRY,rX[15:1]}; CARRY <– rX[0]	?	?
45	**SRX rX**	Shift register rX right. Bit rX[7] is unaffected	rX <– {rX[15],rX[15:1]}; CARRY <- rX[0]	?	?
46	**SUB rX, kk**	Subtract literal kk from register rX	rX <– rX - kk	?	?
47	**SUB rX, rY**	Subtract register rY from register rX	rX <– rX - rY	?	?
48	**SUBC rX, kk**	Subtract literal kk from register rX with CARRY	rX <– rX - kk - CARRY	?	?
49	**SUBC rX, rY**	Subtract register rY from register rX with CARRY	rX <– rX - rY - CARRY	?	?
50	**TEST rX, kk**	Test bits in register rX against literal kk	If (rX & kk) == 0; ZERO <– 1; CARRY <– ODD PARITY (rX & kk)	?	?
51	**TEST rX, rY**	Test bits in register rX against register rY	If (rX & rY) == 0; ZERO <– 1; CARRY <– ODD PARITY (rX & rY)	?	?
52	**XOR rX, kk**	Bitwise XOR register rX with literal kk	rX <– rX ^ kk	?	0
53	**XOR rX, rY**	Bitwise XOR register rX with register rY	rX <– rX ^ rY	?	0
54	**FETCH rX, kk**	Read scratchpad RAM location kk into register rX	rX <- RAM[kk]	-	-
55	**FETCH rX, (rY)**	Read scratchpad RAM pointed to by rY into register rX	rX <- RAM[(rY)]	-	-
56	**STORE rX, kk**	Write register rX to scratchpad RAM location kk	RAM[kk] <- rX	-	-
57	**STORE rX, (rY)**	Write register rX to scratchpad RAM pointed to by rY	RAM[(rY)]<- rX	-	-

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

3. Requirements:

3.1.1 Performance Requirements

The full duplex Universal Asynchronous Receiver will be designed in the Nexy4 DDR FPGA board. It will be able to transmit and receive data via the following baud rates: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 and 921600.

3.1.2 Interface Requirements

The Transmit Engine, Receive Engine, TramelBlaze, TSI, and the serial terminal RealTerm will be used to transmit and receive data.

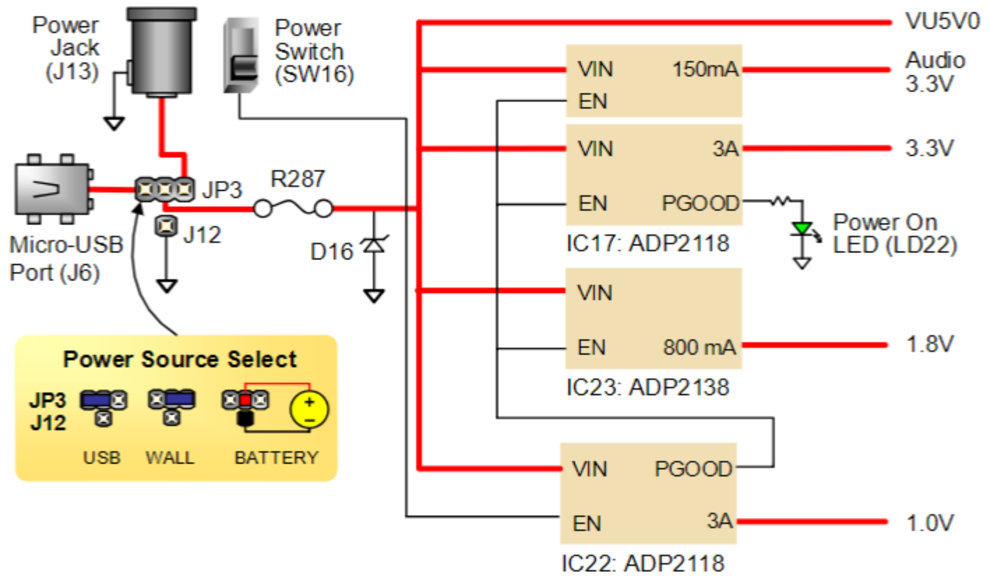
3.1.3 Physical Requirements

The highest baud rate the full duplex UART engine is capable to communicate via the serial terminal without breaking the data being sent is 921600.

3.1.4 Power Requirements

The Nexys4 DDR board can receive power from the Digilent USB-JTAG port(J6) or from an external power supply. Jumper JP3 (near the power jack) determines which source is used. All Nexys4 DDR power supplies can be turned on and off by a single logic-level power switch (SW16).

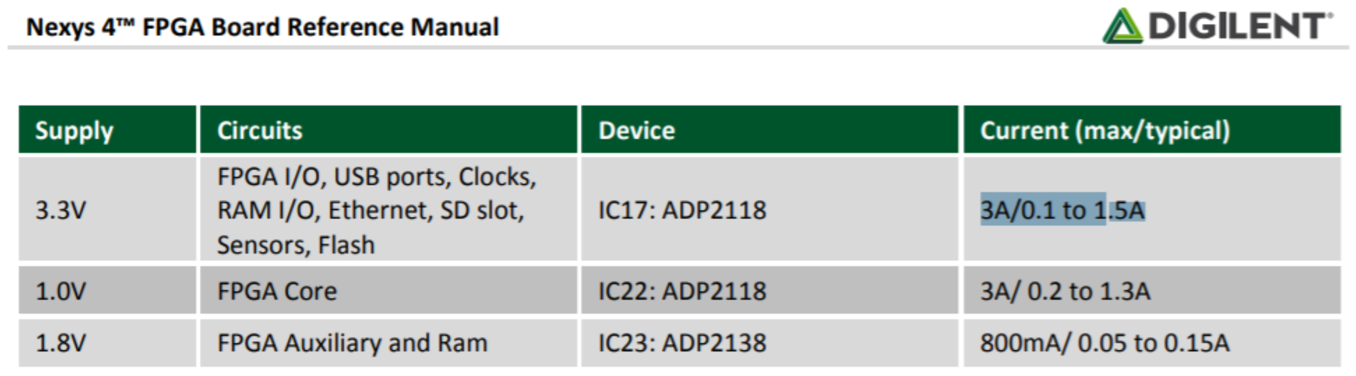
A power-good LED (LD22), driven by the "power good" output of the ADP2118 supply, indicates that the supplies are turned on and operating normally.



The USB port can deliver enough power for the vast majority of designs. The out-of-the-box demo draws ~400mA of current from the 5V input rail. Depending on your application, more power might be required to power your design. Some applications can be run without being connected to a PC's USB port. In these instances, an external power supply or battery pack can be used.

External power supply can be used by plugging into the power jack (JP3) and setting jumper J13 to "wall". The supply must use a coax, center-positive 2.1mm internal-diameter plug and deliver 4.5VDC to 5.5VDC and at least 1A of current.

External battery pack can be used by connecting the battery's positive terminal to the center pin of JP3 and the negative terminal to the pin labeled J12, directly below JP3. Since the main regulator on the Nexys4 DDR cannot accommodate input voltages over 5.5VDC, an external battery pack must be limited to 5.5VDC. If the USB Host function (J5) is used, at least 4.6V needs to be provided. In other cases, the minimum voltage is 3.6V.



Nexys 4™ FPGA Board Reference Manual DIGILENT

Supply	Circuits	Device	Current (max/typical)
3.3V	FPGA I/O, USB ports, Clocks, RAM I/O, Ethernet, SD slot, Sensors, Flash	IC17: ADP2118	3A/0.1 to 1.5A
1.0V	FPGA Core	IC22: ADP2118	3A/ 0.2 to 1.3A
1.8V	FPGA Auxiliary and Ram	IC23: ADP2138	800mA/ 0.05 to 0.15A

Table 2. Nexys 4 Power Supplies

4. Implementation

4.1.1 Design Description

This project implements the Transmit and the Receive Engine with selectable baud rates, the number of bits transmitted, parity enable/disable and odd/even parity bits. For the Transmit Engine to function, the designer needs to implement SR flip flops, an 8-bit loadable register, one-bit register, the parity bit decoder, bit time counter, bit counter and shift register. For the Receive Engine, the designer must implement receive engine control and data path. The goal of the receive engine is to synchronize the data collection with the Tx communication by always polling the Rx line looking for a high to low transition indicating the arrival of the Start bit. A simple finite state machine ensures that the Start bit remains active until the mid-bit at which time the data collection will proceed at a bit time until all bits are received. The FSM will output Start and DoIt indicating that the receive engine is looking for a Start bit and that the receive engine is currently processing data.

The receive engine data path is used to read and write data. There are two addresses dedicated to the UART: 0000 (UART DATA) and (UART Status). When the TramelBlaze writes address 0000, the transmit engine will receive the data byte. When the TramelBlaze reads address 0000 the receive engine will respond with the received data. When the TramelBlaze reads address 0001 it will receive the status collected by the receive engine along with two ready bits (tx ready, rx ready). The other three status flags PERR (Parity Error) indicates that a Parity Error has been detected, FERR (Framing Error) n indicates that the synchronization is lost when no Stop bit is found, and lastly OVF (Overflow) indicates that another byte is being received before the TramelBlaze has read the previous byte.

The first block inside the Tx Engine, is a SR flip flop that is used to set the TxRdy signal high at reset. The second SR flop goes low at reset and both SR flops are used to maintain stable outputs after the inputs are turned off. The 8-bit loadable register that is used to load the data in when the load input pin goes high and outputs 8 bits of data. The 8 bits of data are then used to determine how many bits are needed to produce a transmission of data. The user determines what type of protocol to send out. The are 7 options to transmit data from the inputs eight, pen (parity enable), and ohel (odd and even bits): 7N1, 7E1, 7O1, 8N1, 8E1, and 8O1. The truth table is used to produce the combo logic circuit that produces the output bits 10 and 9.

Eight	Parity Enable	Odd and Even	Bit 10	Bit 9
0	0	0	1	1
0	0	1	1	1
0	1	0	1	EP
0	1	1	1	OP
1	0	0	1	D7
1	0	1	1	D7
1	1	0	EP	D7
1	1	1	OP	D7

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

At the heart of the Transmit Engine, is a 11-bit shift register that is used to shift the transmitted data one bit of a time. Once the parity decoder determines bit 10 and bit 9, a one-bit register goes high, it tells the shift register to load the data to bit-10, bit-9, bits 8-2 and bit-1 goes high and bit-0 goes low. If the reset is pressed, the shift registers outputs 11-bits 1's. The output of the shift register gets the LSB 0 of the data being shifted.

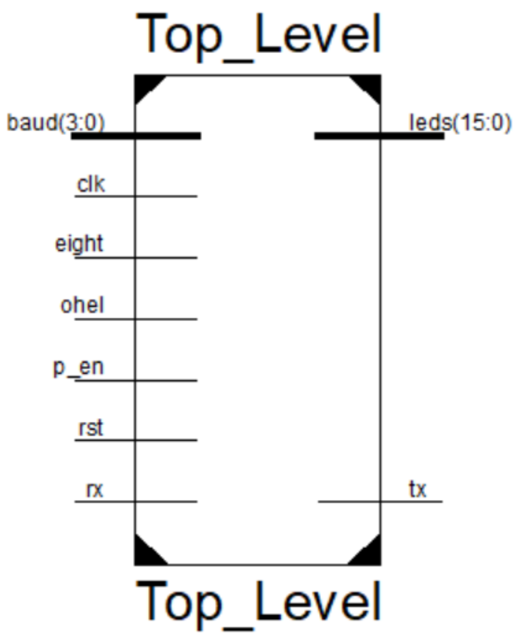
The design must also include two counters: 1) Count clocks to determine the bit time and 2) Count the bits to know when we are done. These two counters will let us know when we have waited a bit time and to know when all the bits have been transmitted. Lastly, a Baud Decoder circuit is created to generate the number of clocks that are necessary to fill one-bit time. Combining all the modules describe above, produces the Transmit Engine. Once the Transmit and Receive Engine are completed, you can interface the UART with the 16-bit TramelBlaze microcontroller. For the user to select a different baud speed, the following table is used to select the baud by using the on-board switches.

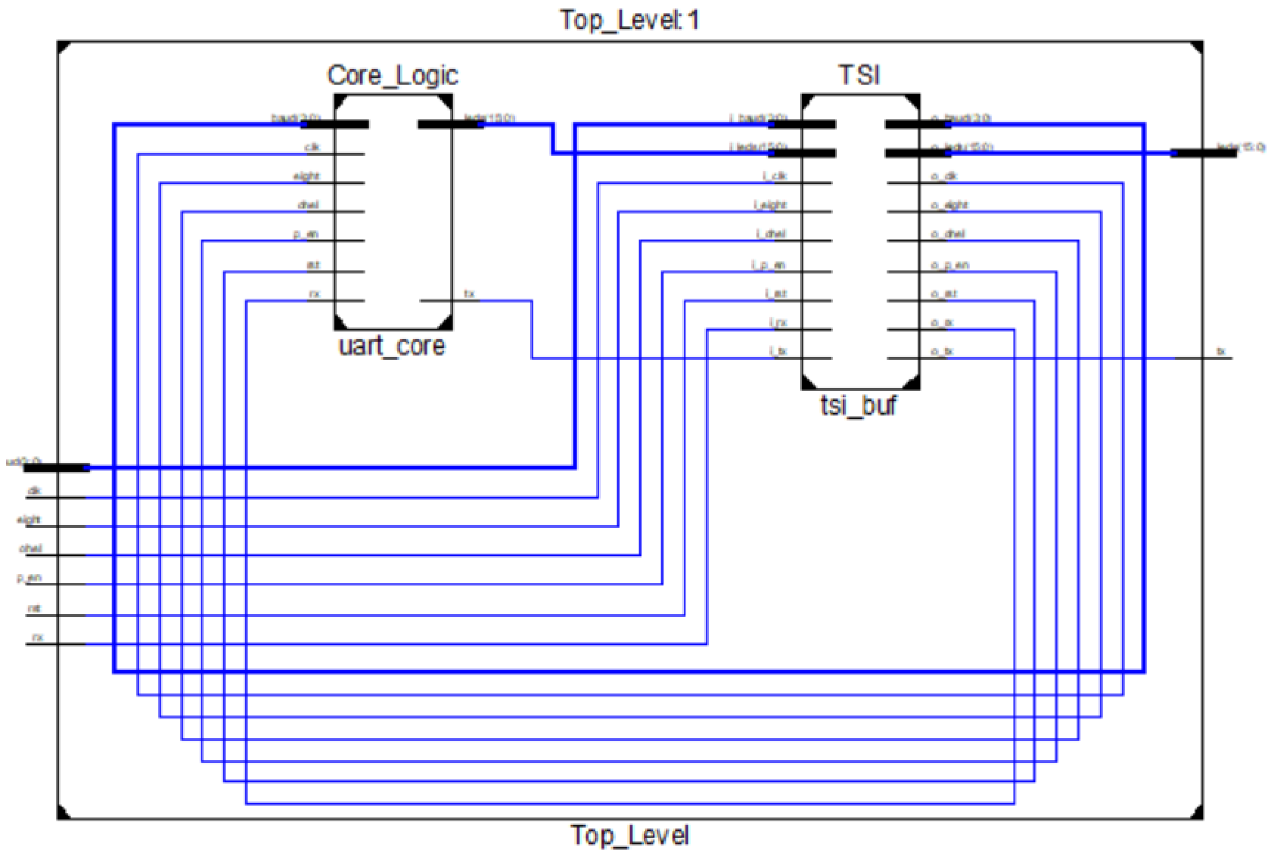
Baud Switches	Baud Rate	Bit Time	Nexys4 Count
0000	300	0.003333	333,333
0001	1200	0.0008333	83,333
0010	2400	0.000416667	41,667
0011	4800	0.000208333	20,833
0100	9600	0.000104167	10,417
0101	19200	5.20833E-05	5,208
0110	38400	2.60417E-05	2,604
0111	57600	1.73611E-05	1,736
1000	115200	8.68056E-05	868
1001	230400	4.34028E-06	434
1010	460800	2.17014E-06	217
1011	921600	1.08507E-06	109

4.1.2 Top Level Description

The top-level block diagram contains the Core design and the TSI. The Core design consist of the full UART and TSI contains all the references to the target technology libraries. For the Core design to communicate with the outside world, it's I/O's must pass through the TSI before interacting with the FPGA. The assembly code written for the TramelBlaze will be used to display text to the serial terminal. The requirements are to display a banner, hometown when entering an asterisk, line count, and backspace to delete previously entered characters.

4.1.3 Top Level Block Diagram





I/O		**Signal Description**	**FPGA board Switches & Buttons**
Input	clk	100MHz FPGA board crystal oscillator	None
Input	rst	Synchronous Rest	
Input	baud	4-bit input used to select baud rate	LOC Pin Assignment: R13(SW8), U18(SW7), T18(SW6), R17(SW5)
Input	eight	1-bit input to select [7:0] ot [6:0] bits	LOC Pin Assignment: R15(SW4)
Input	p en	Parity enable – 1: odd parity, 0: no parity	LOC Pin Assignment: M13(SW3)
Input	ohel	odd high even low- 1: odd parity, 0: even parity	LOC Pin Assignment: L16(SW2)
Input	rx	Receive signal which indicates a high to low transition	LOC Pin Assignment: C4
Output	tx	Transmission signal which indicates that data is being shifted out from Transmit Engine	LOC Pin Assignment: D4

4.1.5 Clocks

All blocks in the design used one clock that runs at a frequency of 100MHz and period of 10 nanoseconds.

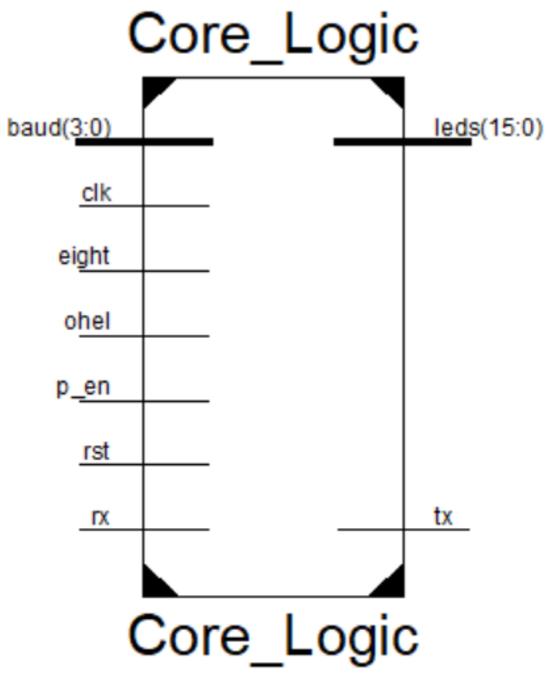
4.1.6 Resets

All the designs used a synchronous reset (AISO) to prevent metastability. The purpose for the AISO circuit in our design is to prevent any of the flops in our design from entering a metastable state when reset is released. Synchronously releasing reset ensures all-of the flops are reset at the same time and that they are stable when reset is released.

5. Internally Acquired Blocks

5.1 Core Logic

The Core Logic Circuit is used to combine the Address Decoder, UART, and TramelBlaze so it could interface with the technology specific instantiation (TSI) circuit. The core logic also contains and AISO, PED, SR flop, counters, load register, and shift register that are essential to the design.

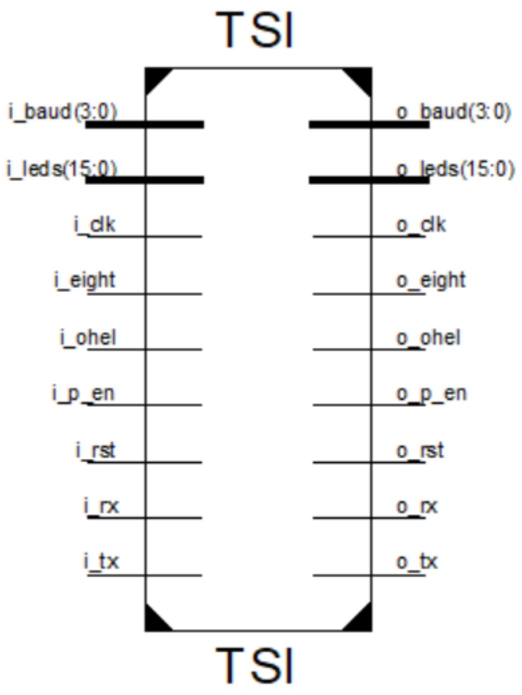


Name	I/O	Description
clk	I	Synchronous clock
rst	I	Synchronous reset
baud[3:0]	I	4-bit input to select baud rate
eight	I	1-bit input to select 8-bit data transmission
ohel	I	1-bit input to select if we want even or odd
p_en	I	1-bit input to select parity bit
rx	I	Receive signal looks for a high to low transition indicating the arrival of the start bit
leds	O	LEDs are used to demonstrate the TramelBlaze is reading data
tx	O	Data being shifted out of the transmit engine

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

5.2 Technology Specific Instantiation (TSI)

The TSI contains all references to the target technology library. All communications to or from the I/O of the device pass through the TSI. Each I/O of the chip must have a particular selected device to meet electrical timing requirements of the external interface.



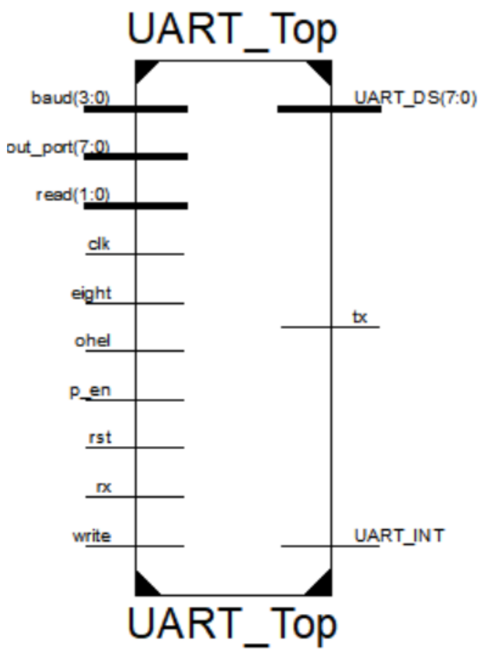
Name	I/O	Description
i_baud[3:0]	I	Baud input buffer
i_leds	I	LEDs input buffer
i_clk	I	Clock input buffer
i_eight	I	Eight input buffer
i_ohel	I	Odd high even low input buffer
i_p_en	I	Parity enable input buffer
i_rst	I	Reset input buffer
i_rx	I	Receive input buffer
i_tx	I	Transmit input buffer
o_baud[3:0]	O	Baud output buffer
o_leds	O	LEDS output buffer
o_clk	O	Clock output buffer
o_eight	O	Eight output buffer
o_ohel	O	Odd high even low output buffer
o_p_en	O	Parity enable buffer
o_rst	O	Rest output buffer

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

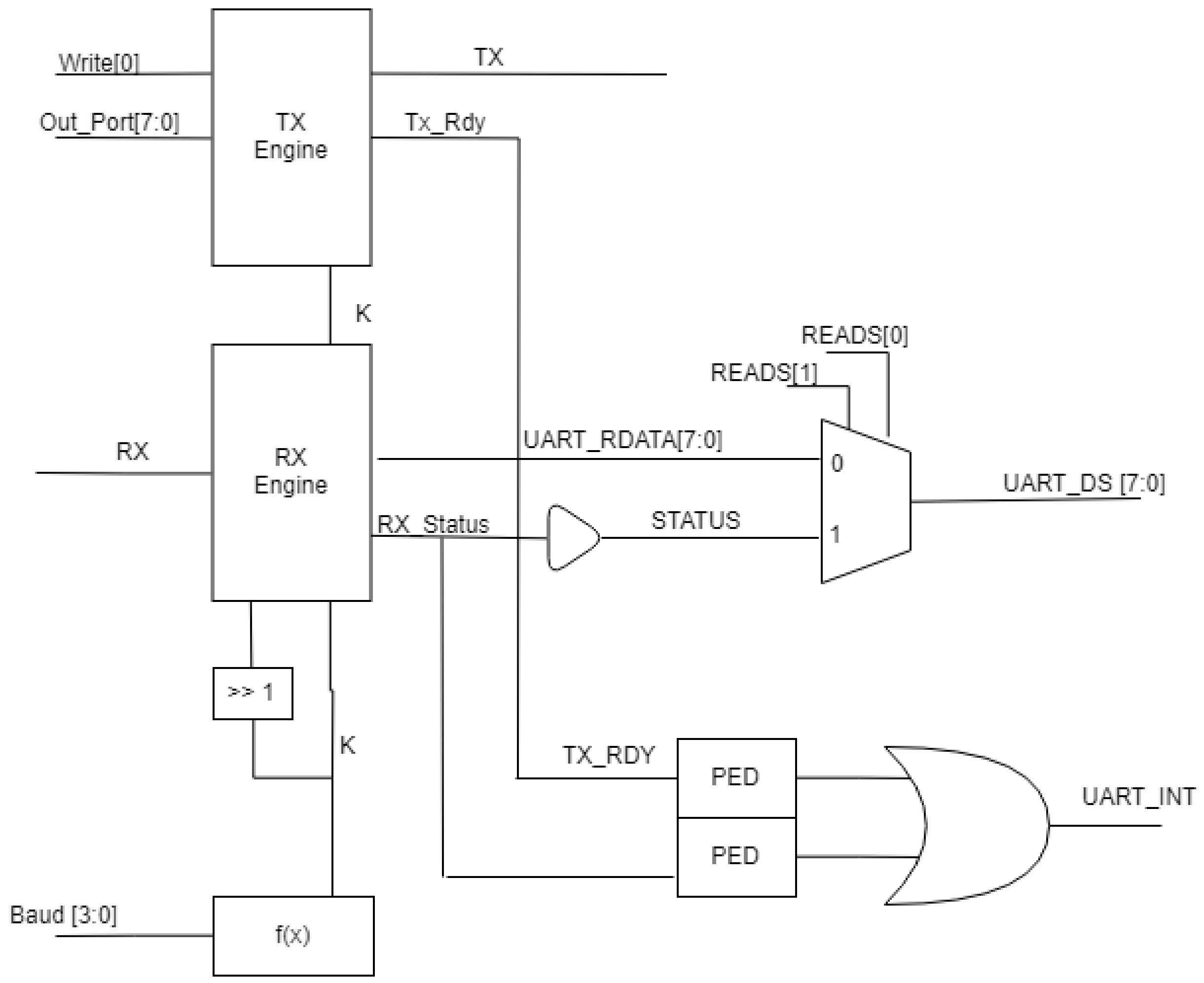
o_rx	O	Receive output buffer
o_tx	O	Transmit output buffer

5.3 UART Top

The UART Top in composed of the Transmit Engine, Receive Engine, Baud Decoder, UART status flags and pulse edge detectors. Both engines in this design require a specific baud rate selected by the user. An interrupt signal is also implemented to cause the TramelBlaze to enter an Interrupt Service Routine. The UART top also contains status flag such as parity error, over flow, and framing error. The tx signal is used to transmit data over USB.



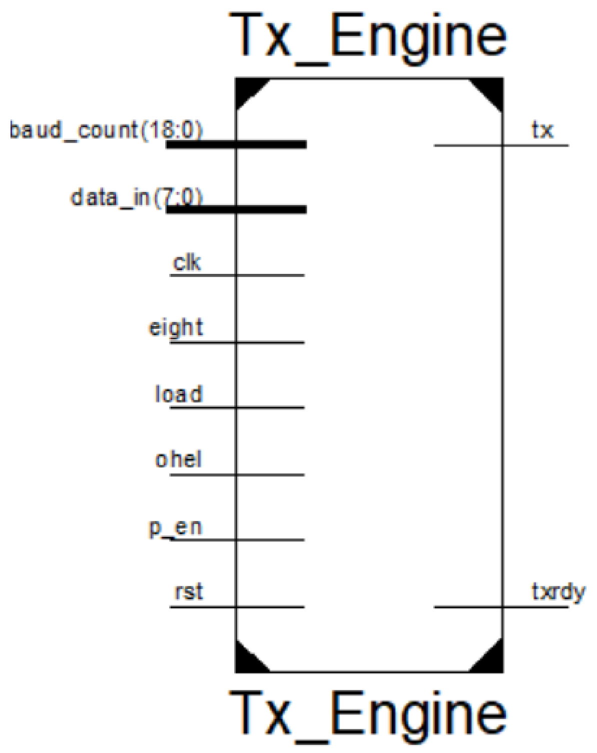
Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1



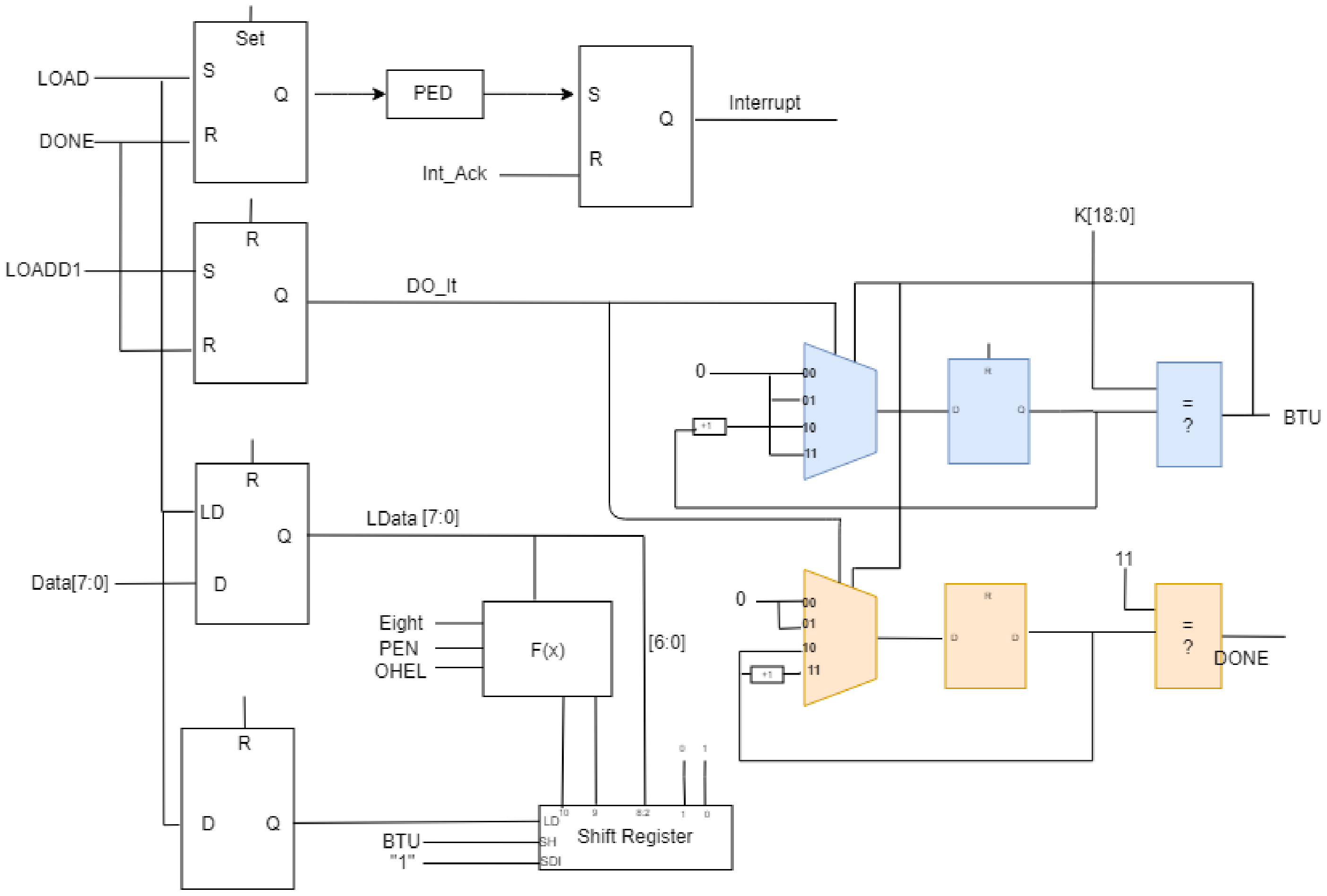
Name	**Signal Description**	**I/O**	**FPGA board Switches & Buttons**
clk	100MHz FPGA board crystal oscillator	I	None
rst	Synchronous Rest	I	LOC Pin Assignment: M18
write	Write address 0000	I	None
read	Read address 0001	I	None
out_port	8-bit data coming from TramelBlaze	I	None
baud	4-bit input used to select baud rate	I	LOC Pin Assignment: R13(SW8), U18(SW7), T18(SW6), R17(SW5)
eight	1-bit input to select [7:0] ot [6:0] bits	I	LOC Pin Assignment: R15(SW4)
p_en	Parity enable – 1: odd parity, 0: no parity	I	LOC Pin Assignment: M13(SW3)
ohel	odd high even low- 1: odd parity, 0: even parity	I	LOC Pin Assignment: L16(SW2)
rx	Receive signal which indicates a high to low transition	I	LOC Pin Assignment: C4
tx	Transmission signal which indicates that data is being shifted out from Transmit Engine	O	LOC Pin Assignment: D4
UART_DS	Status flags: parity error, framing error, and over flow error	O	None
UART_Int	Set an interrupt signal that gets feed into the TramelBlaze	O	None

5.4 Transmit Engine

The Transmit Engine is composed of an SR flop, loadable register, parity decoder, a register, shift register, bit time counter, and a bit counter.



Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

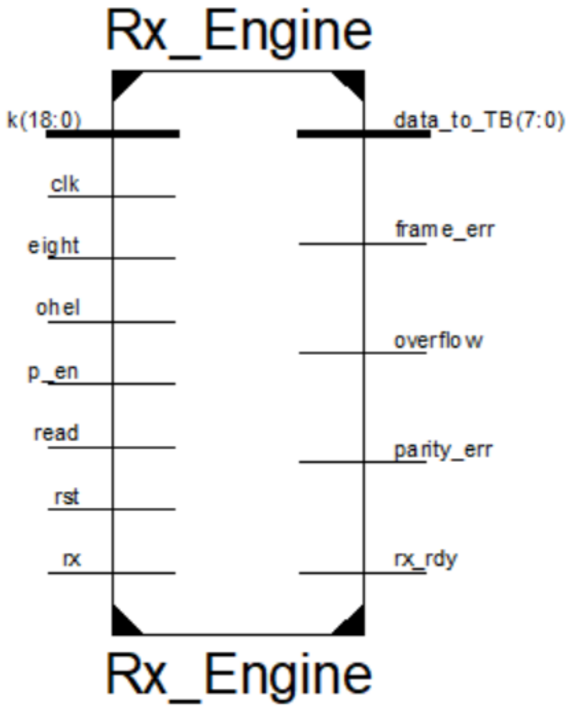


Name	I/O	Description
baud_count[18:0]	I	Baud Rate Decoder
data_in[7:0]	I	Data from TramelBlaze
clk	I	100MHz crystal oscillator
eight	I	8 bits
ohel	I	Odd high even low
p_en	I	Parity enable
load	I	From Address Decoder
rst	I	AISO
tx	O	USB
tx rdy	O	UART Interrupt

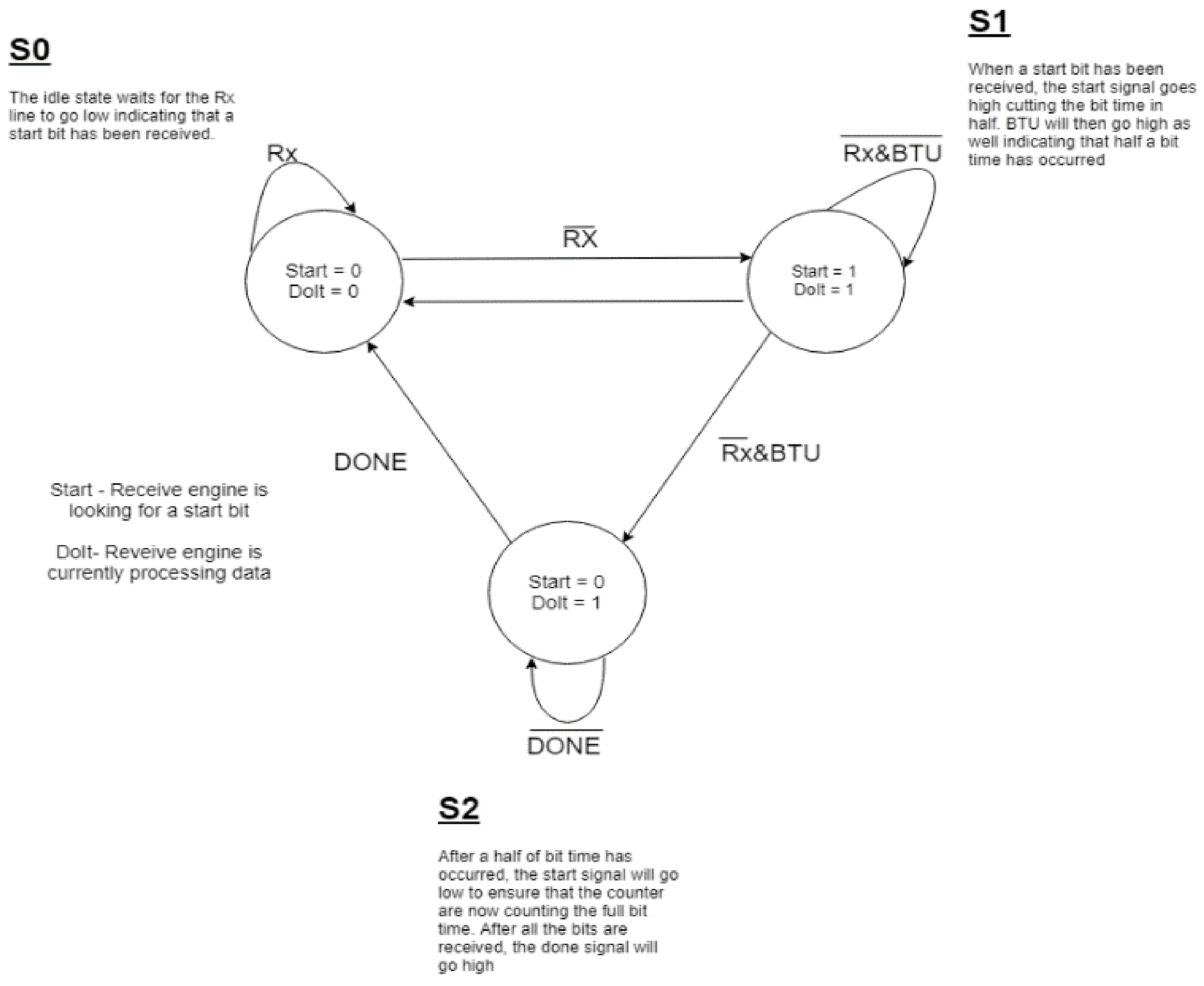
Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

5.5 Receive Engine

The Receive Engine is composed of control and data path. The control path contains a finite state machine, bit time counter and bit counter to sync the signals coming into the UART. The data path checks for the correct transmission, if the transmission is incorrect the status flags are set to indicate which type of error it was. Otherwise, the data will be outputted to the TramelBlaze.

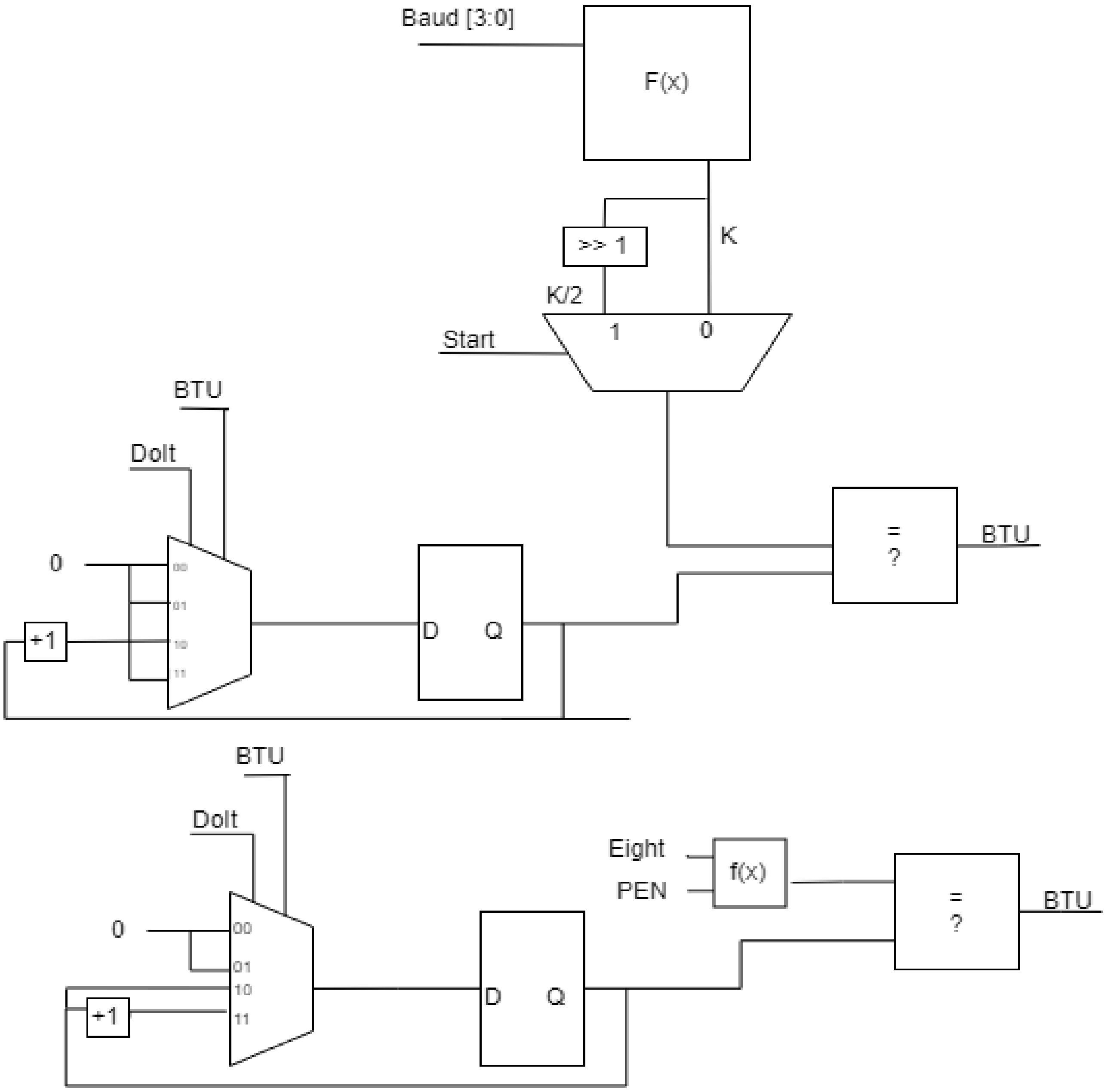


Name	I/O	Description
k[18:0]	I	Baud Rate Decoder
clk	I	100MHz crystal oscillator
eight	I	On board switch
ohel	I	On board switch
p en	I	On board switch
read	I	TramelBlaze
rst	I	AISO
rx	I	UART Interrupt
data to TB[7:0]	O	TramelBlaze
frame err	O	Framing error
over flow	O	Overflow error
parity err	O	Parity error
rx rdy	O	TramelBlaze

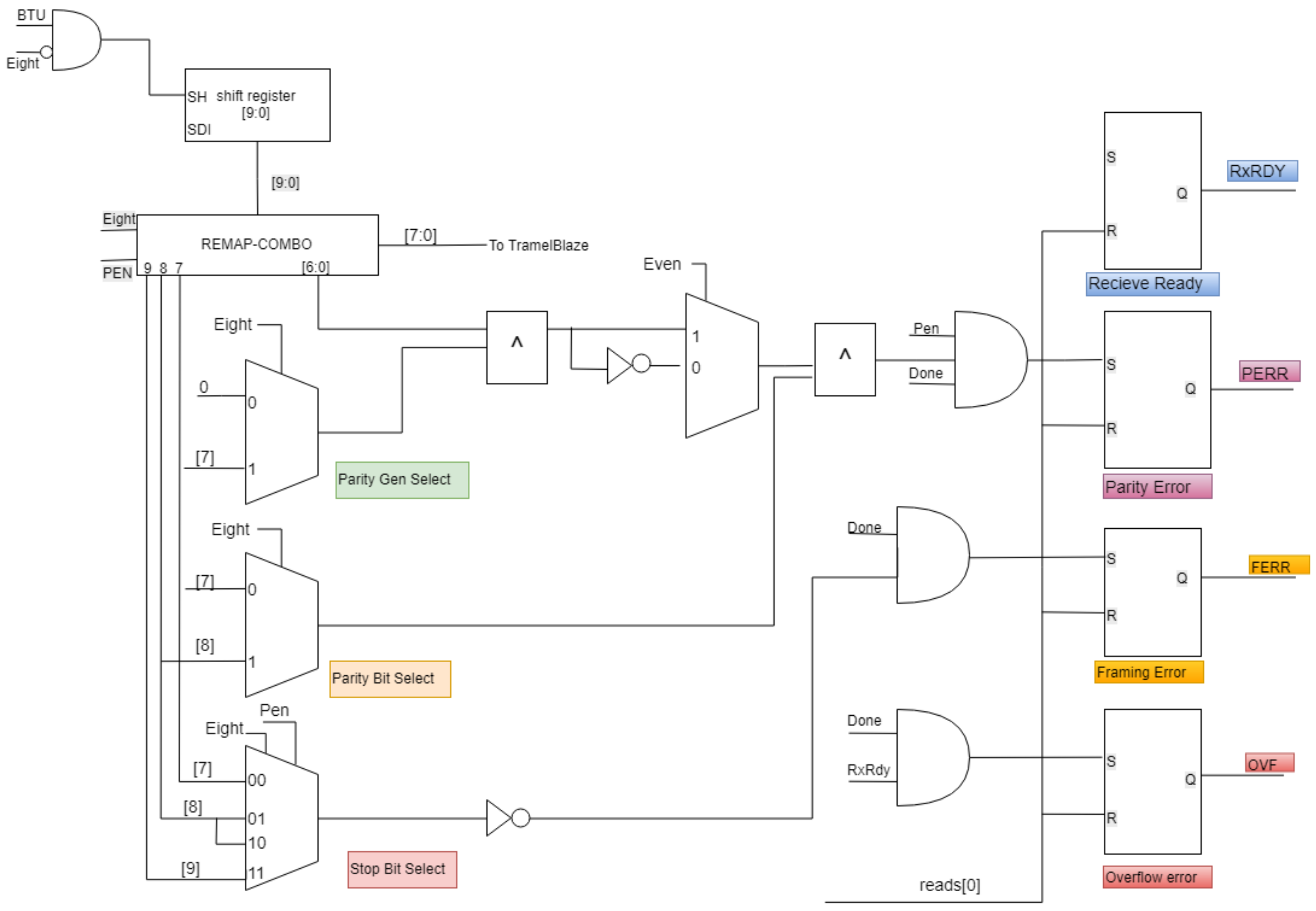


S0
The idle state waits for the Rx line to go low indicating that a start bit has been received.
S1
When a start bit has been received, the start signal goes high cutting the bit time in half. BTU will then go high as well indicating that half a bit time has occurred
Rx
$\overline{Rx\&BTU}$
$\overline{RX}$
Start = 0
DoIt = 0
Start = 1
DoIt = 1
DONE
$\overline{Rx}$&BTU
Start - Receive engine is looking for a start bit
DoIt- Reveive engine is currently processing data
Start = 0
DoIt = 1
$\overline{DONE}$
S2
After a half of bit time has occurred, the start signal will go low to ensure that the counter are now counting the full bit time. After all the bits are received, the done signal will go high

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

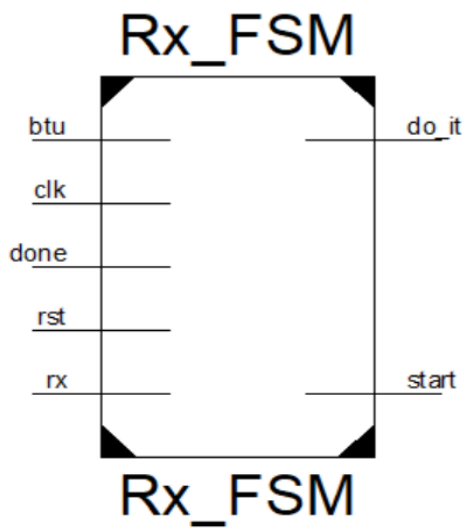


Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1



5.6 Rx FSM

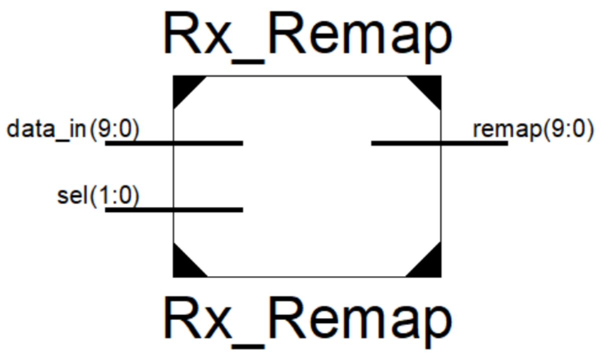
The Receive Engine contains a finite state machine that determines if its receiving a signal or not.



Name	I/O	Description
btu	I	Rx bit time counter
clk	I	100MHz crystal oscillator
done	I	Rx bit counter
rst	I	AISO
rx	I	USB
do it	I	Rx bit time counter and bit counter
start	I	Rx bit time counter

5.7 Rx_Remap

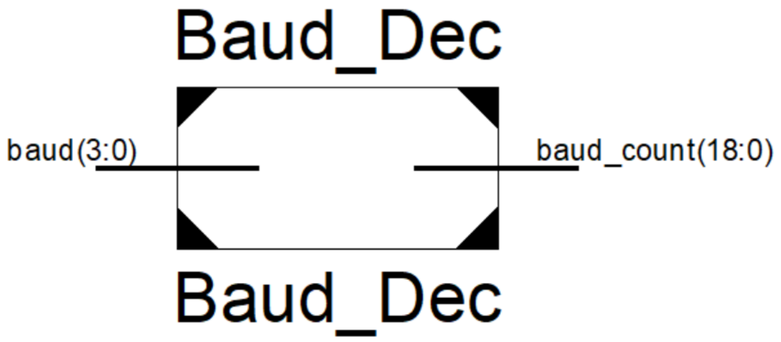
The purpose for the Rx ramap circuit is to justify the received data. Depending on the transmission data, the remap will adjust the data according to the eight and parity enable select switches. 8-bits from the remap circuit will go to the TramelBlaze, while the remaining bits will be distributed accordingly.



Name	I/O	Description
data in[9:0]	I	Data coming in from shift register
sel[1:0]	I	Eight and Parity enable inputs
remap[9:0]	O	Data out

5.8 Baud Decoder

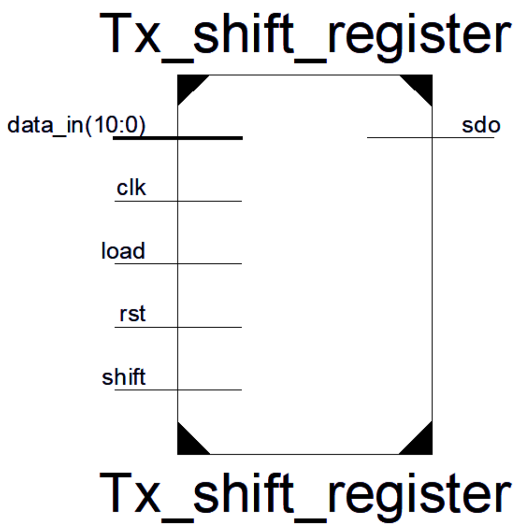
The Baud decoder is a mux that selects how many bits per second to run the design.



Name	I/O	Description
baud[3:0]	I	On board switches
baud count	O	Transmit Engine

5.9 Tx Shift Register

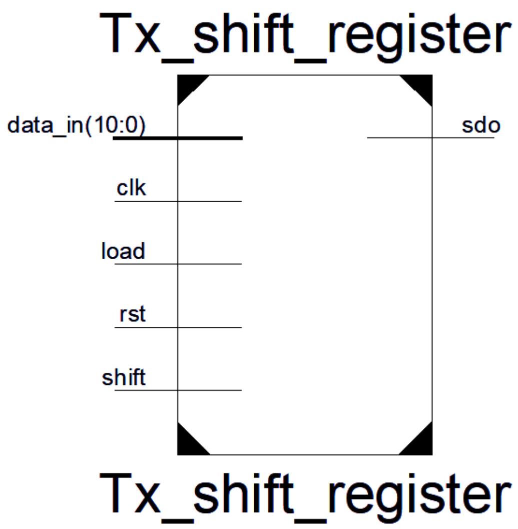
The shift register is responsible for taking in the data and shifting them into a register. If the switches eight and parity enable are selected, the data must be remapped accordingly.



Name	I/O	Description
clk	I	100MHz crystal oscillator
rst	I	AISO
load	I	Load
shift	I	Shift Data
data in[10:0]	I	Data coming in
sdo	O	Data shifted out

5.10 Tx Bit Time Counter

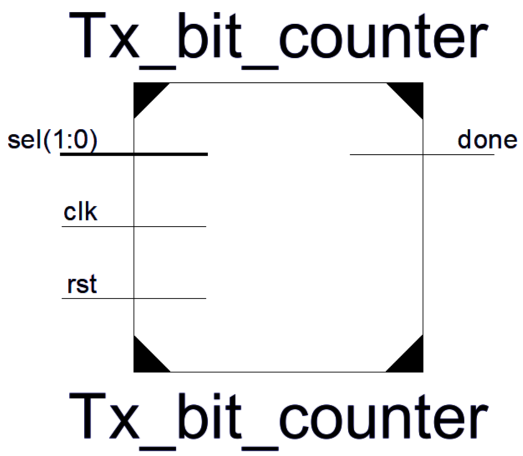
The Bit Time Counter is used to determine the speed at which the engine runs caused by the value selected in the baud decoder.



Name	I/O	Description
baud[3:0]	I	On board switches
sel[1:0]	I	DoIt and BTU
clk	I	100MHz crystal oscillator
btu	O	Tx shift register and bit count

5.11 Tx Bit Time Counter

The Bit Counter is used to count the number of bits to ensure that the engine is picking up the correct number of bits. It also checks the values to see if there is any discrepancies by seen how many bits are need to receive based on the eight and parity enable switches.

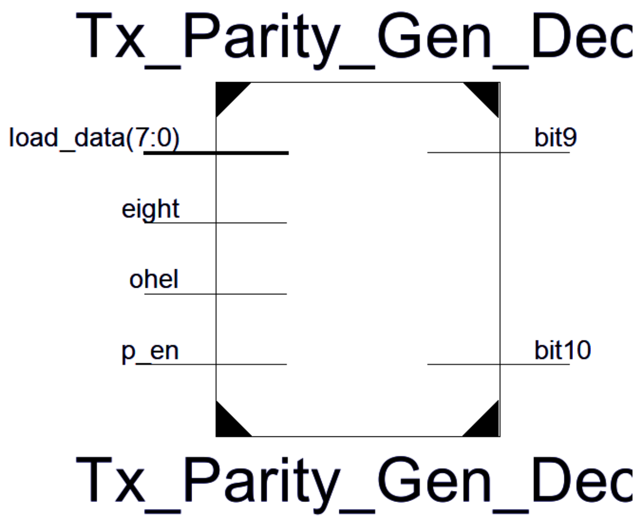


Name	I/O	Description
sel[1:0]	I	DoIt and BTU
clk	I	100MHz crystal oscillator
rst	I	AISO

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

5.12 Tx Parity Generator Decoder

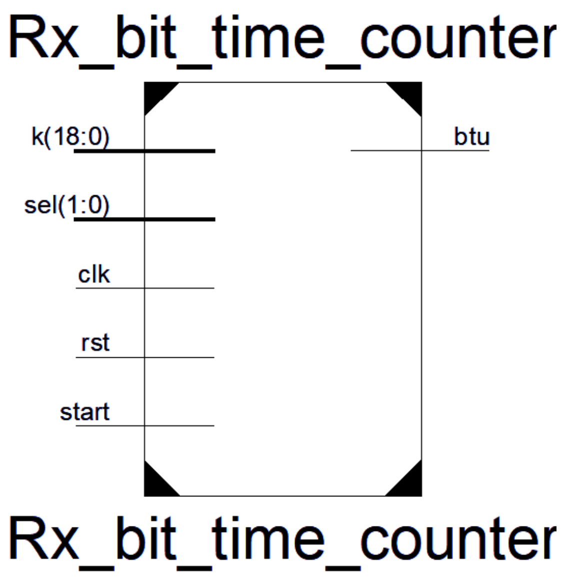
The transmit parity generator decoder is responsible for generating the even or odd parity bits depending on the configuration. If eight is selected, the loadable register will load 8 bits, else 7-bits. Depending on the data transmitted, if the data transmitted requires a parity bit the data will be account for and an even or odd no parity bit will be added.



Name	I/O	Description
load data[7:0]	I	8 bit data coming loadable register
eight	I	On board switch
ohel	I	On board switch
p_en	I	On board switch
bit9	O	9-bit data
bit10	O	10-bit data

5.13 Rx Bit Time Counter

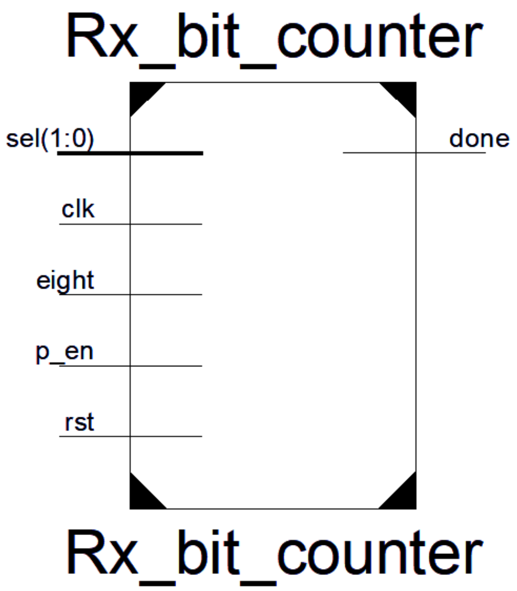
The bit timer counter is responsible for counting the number of clocks depending on the baud rate that is selected. The baud rate is configurable with the on-board switches. The baud rate determines the bit time which is the length of time that a transmitted bit is held on the wire.



Name	I/O	Description
k	I	Baud rate decoder
sel[1:0]	I	DoIt and Btu
clk	I	100MHz clock oscillator
rst	I	AISO
start	I	FSM
btu	O	register

5.14 Rx Bit Counter

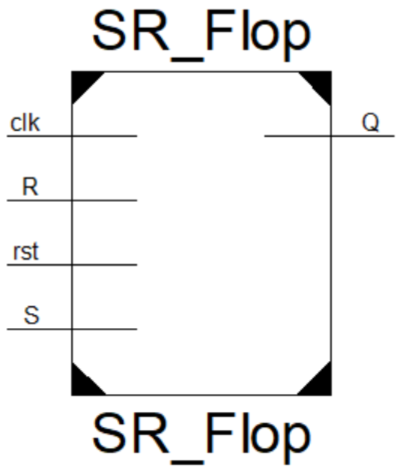
The bit counter in the transmit engine is responsible for counting the number of bits that have been transmitted. The purpose of this block is to count the number of bits that have been transmitted. The number of bits transmitted is always 11 and once all 11-bits have been transmitted, the done signal goes high which set the tx_rdy signal high which indicates that the transmit engine is ready to transmit another byte of data.



Name	I/O	Description
sel[1:0]	I	DoIt and Btu
clk	I	100MHz clock oscillator
eight	I	On board switches
p en	I	On board switches
rst	I	AISO
done	O	FSM and Rx Controller

5.15 SR Flop

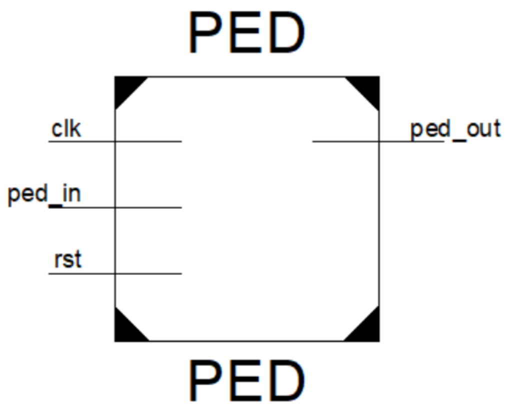
The SR flop are used to store information after the inputs are turned off. The set input causes the output of 1 and the R input cause the output of 0. For the transmit engine, when we reset the design, we need to set the reset value high.



Name	I/O	Description
clk	I	100MHz crystal oscillator
R	I	TramelBlaze
rst	I	AISO
S	I	PED
Q	O	TramelBlaze

5.13 Pulse Edge Detector (PED)

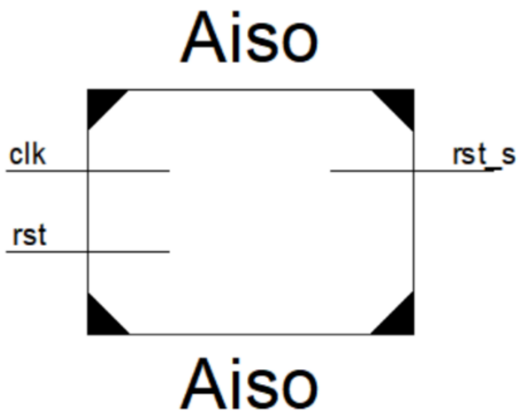
The PED is used to generate a one clock period pulse whenever the signal is asserted.



Name	I/O	Description
clk	I	100MHz crystal oscillator
ped in	I	UART
rst	I	AISO
ped out	O	SR flop

5.14 Asynchronous-In-Synchronous-Out (AISO)

The Asynchronous-In-Synchronous-Out circuit is used to synchronize the reset of digital flip flops.



Name	I/O	Description
clk	I	100MHz crystal oscillator
rst	I	Button
rst s	O	Output to every block

6. Externally Acquired Blocks

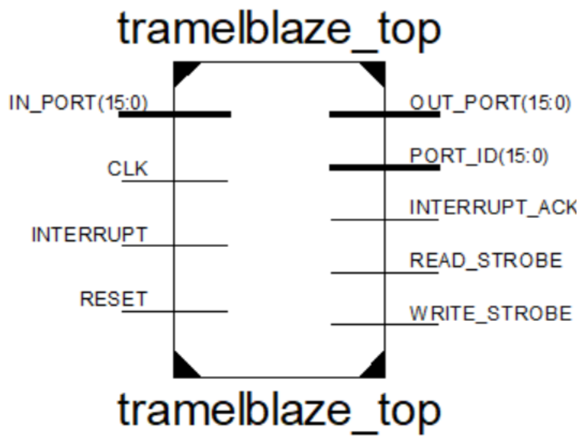
6.1 TramelBlaze

The TramelBlaze is a 16-bit embedded microcontroller designed to emulate the 8-bit PicoBlaze.

6.2 Description

The TramelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze. The instruction set is the PicoBlaze instruction set. It contains three memories Instructions Rom, Call/Return Stack and Scratchpad RAM. The TramelBlaze's I/O are a 16-bit in_Port, Interrupt, Reset, Clk, a 16-bit Out_Port, 16-bit Port_Id output, Read_Strobe output, Write_Strobe output and interrupt acknowledge output. When implementing the TramelBlaze into your design, you must generate the tb_rom and load the coe file into RAM. A Python assembler is used to generate assembly code. We will be using assembly code to output to the serial terminal and keep track of the line count.

6.3 TramelBlaze Block Diagram



6.4 I/O Definitions

The I/O ports extend the TramelBlaze MCU's capabilities and allow the MCU to connect custom peripherals or to FPGA. It contains a 16-bit in_Port input that is used to write to the scratchpad memory. It allows to use 16 16-byte-wide registers. It also handles the interrupt input used to let the MCU that an event occurred and executes interrupt service routine corresponding to the received interrupt. Once the interrupt is complete, it returns to the main program. The TramelBlaze shares the 100MHz clock from the FGPA. The 16-bit output Port_ID produces the address/instruction. The 16-bit Out_Port produces data contained within the TramelBlaze. Read_Strobe output is asserted high indicating that the input data on the in_Port was captured to the specified data register during an input instruction. The Write_Strobe is asserted high validating the output signal data of the Out_Port. Lastly, the Int_Ack is asserted high to acknowledge that an interrupt event occurred.

6.5 I/O Timing

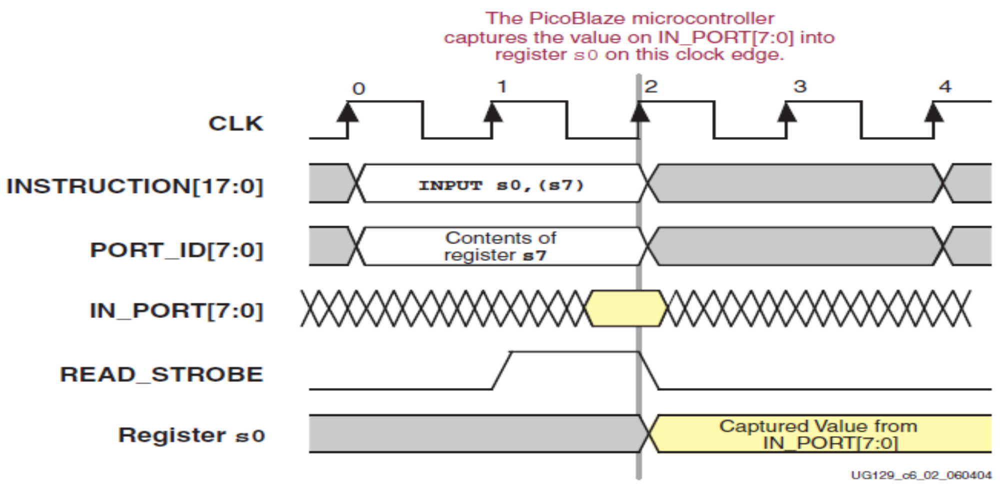
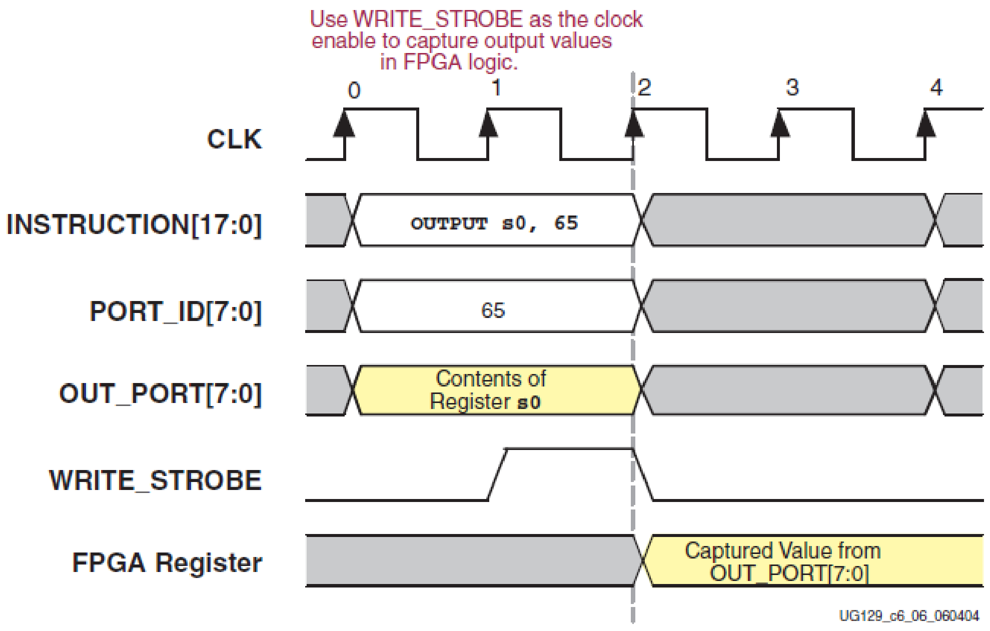

The PicoBlaze microcontroller captures the value on IN_PORT[7:0] into register s0 on this clock edge.
0
1
2
3
4
CLK
INSTRUCTION[17:0]
INPUT s0, (s7)
PORT_ID[7:0]
Contents of register s7
IN_PORT[7:0]
READ_STROBE
Register s0
Captured Value from IN_PORT[7:0]
UG129_c6_02_060404

Figure 6-2: **Port Timing for INPUT Instruction**



\

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

7. Chip Verification

7.1 TestbenchWaveforms

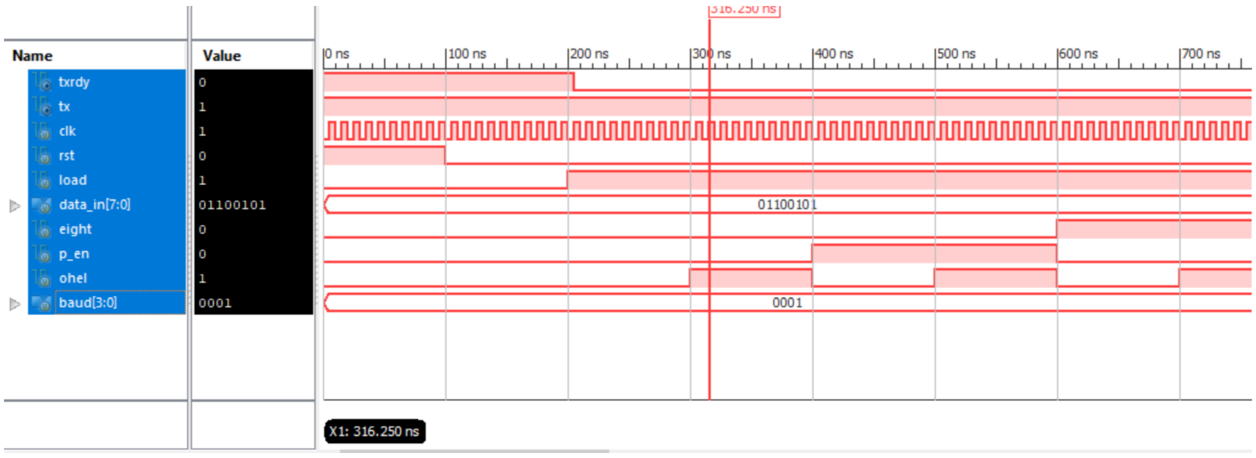


Figure 27: Transmit Engine TestFixture

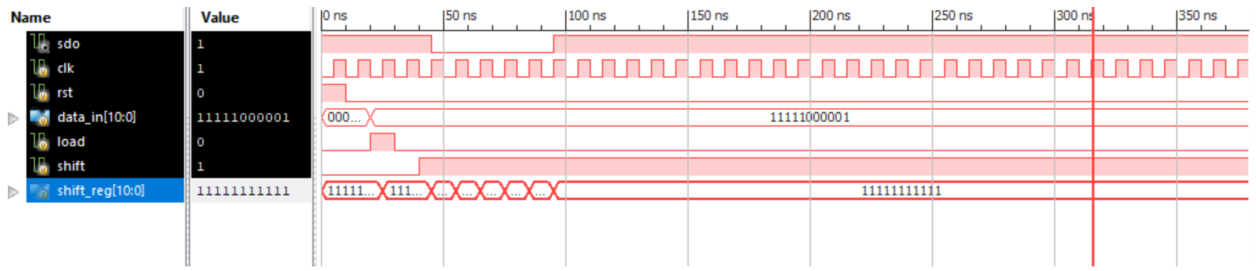


Figure 28: Shift Register TestFixture

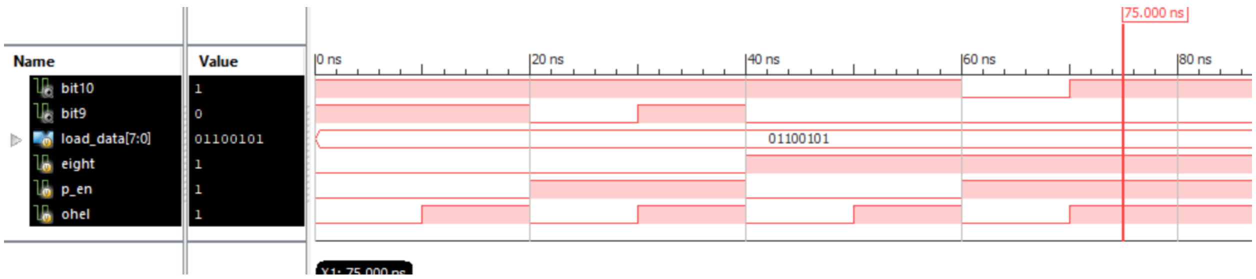
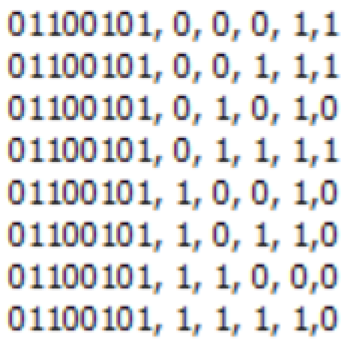


Figure 29: Parity Decoder TestFixture

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1



01100101, 0, 0, 0, 1,1
01100101, 0, 0, 1, 1,1
01100101, 0, 1, 0, 1,0
01100101, 0, 1, 1, 1,1
01100101, 1, 0, 0, 1,0
01100101, 1, 0, 1, 1,0
01100101, 1, 1, 0, 0,0
01100101, 1, 1, 1, 1,0

To the left truth table Inputs: Data_In, Eight, Parity Enable and Ohel. Outputs: bit10 and bit 9 on the right.

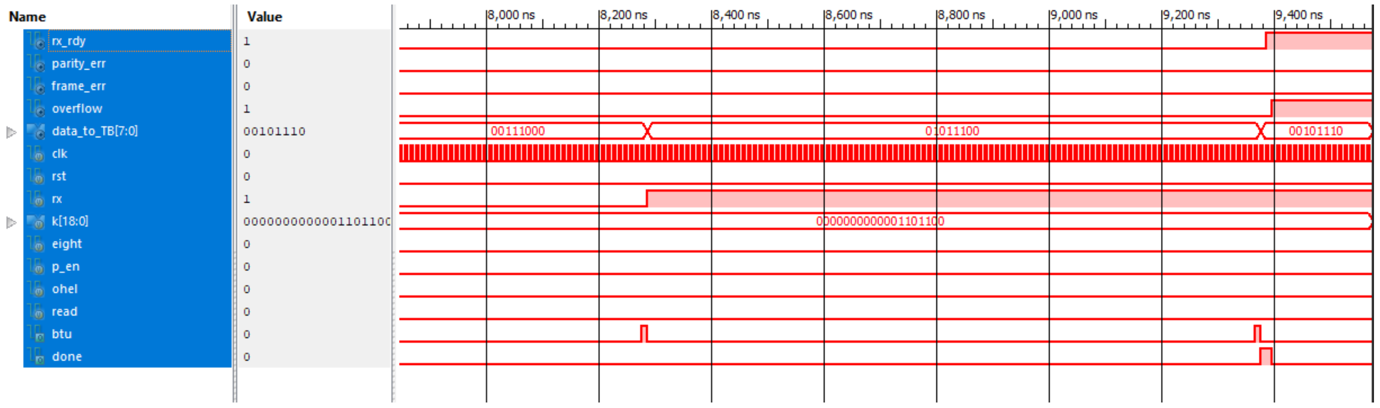


Figure 30: Receive Engine TestFixture

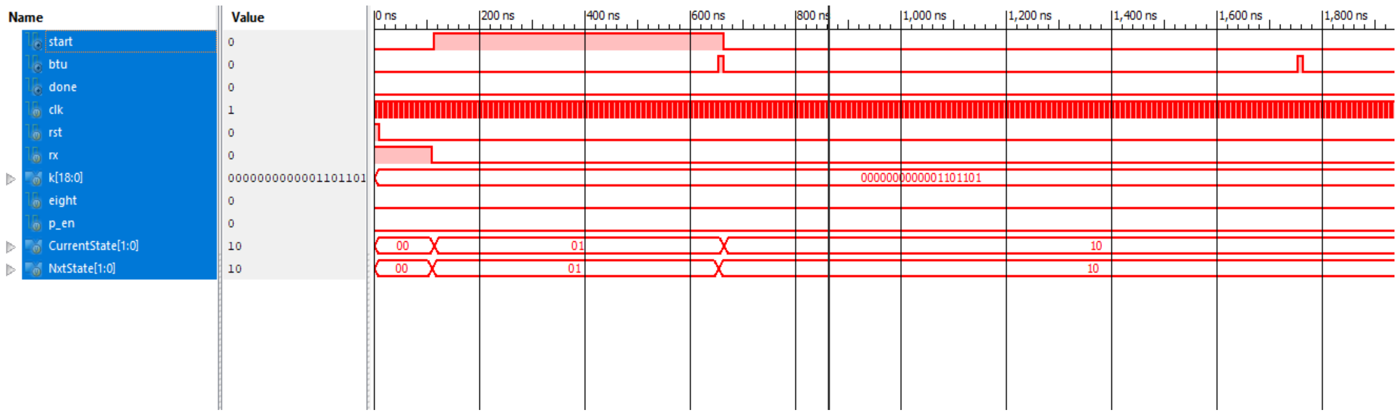
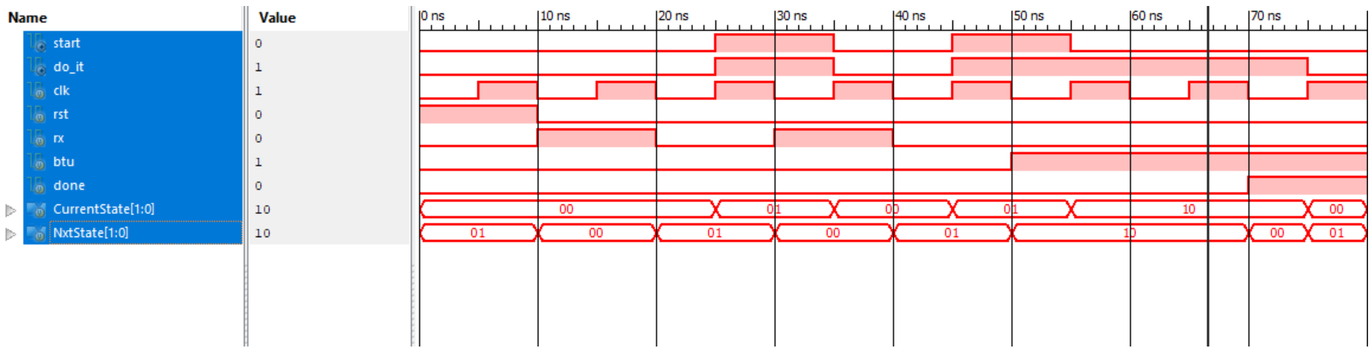


Figure 31: Receive Engine Control Path TestFixture



```
ISim>
# restart
ISim>
# run all
Simulator is doing circuit initialization process.
S0
Finished circuit initialization process.
S0
S1
S0
S1
S2
S2
S0
```

Figure 32: Receive Engine Finite State Machine TestFixture

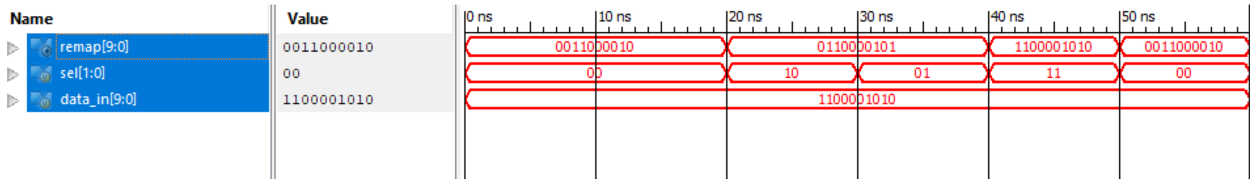


Figure 33: Receive Engine Remap TestFixture

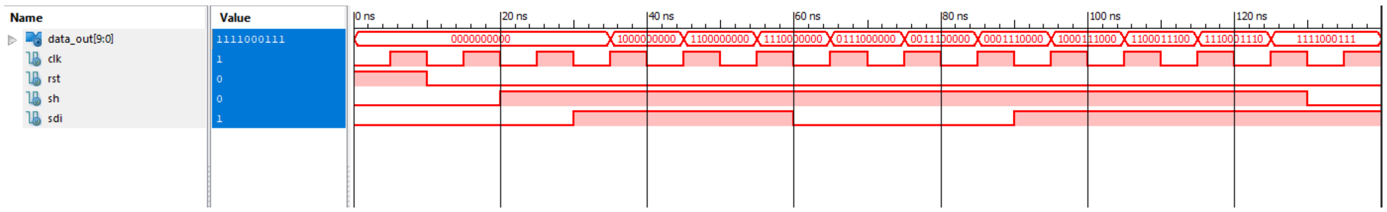
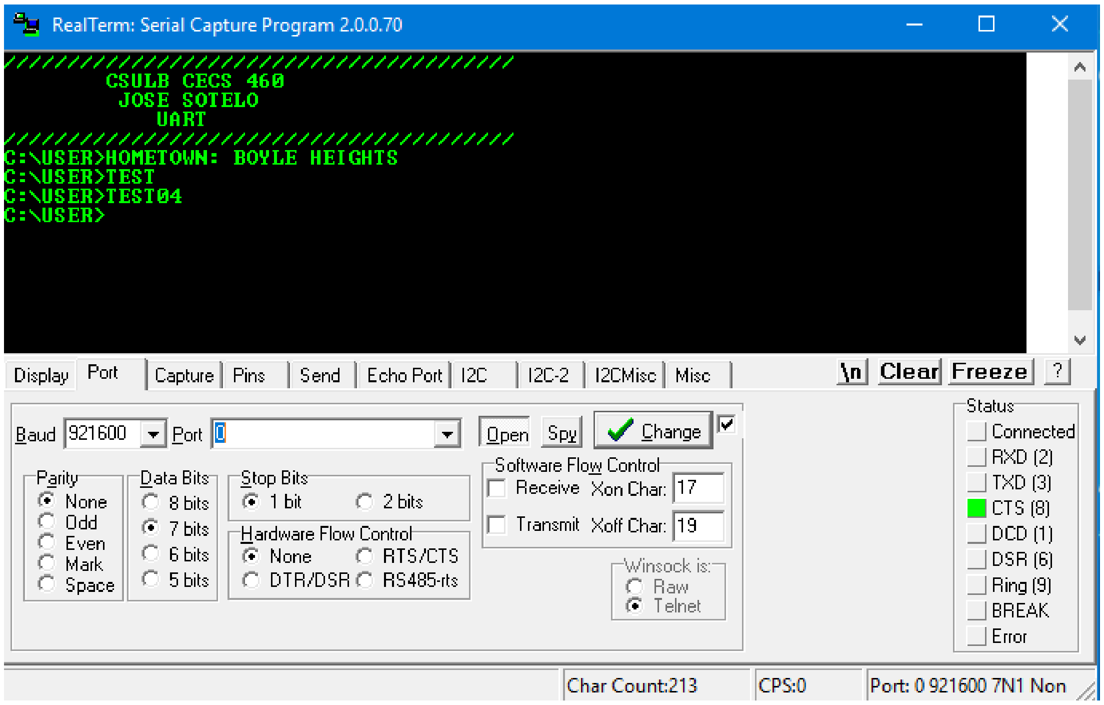


Figure 34: Receive Engine Shift Register TestFixture

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1



RealTerm: Serial Capture Program 2.0.0.70

```
////////////////////////////////////////
        CSULB CECS 460
         JOSE SOTELO
            UART
////////////////////////////////////////
C:\USER>HOMETOWN: BOYLE HEIGHTS
C:\USER>TEST
C:\USER>TEST04
C:\USER>
```

Baud 921600 Port 0

Parity: [✓ None] Odd Even Mark Space

Data Bits: 8 bits [✓ 7 bits] 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

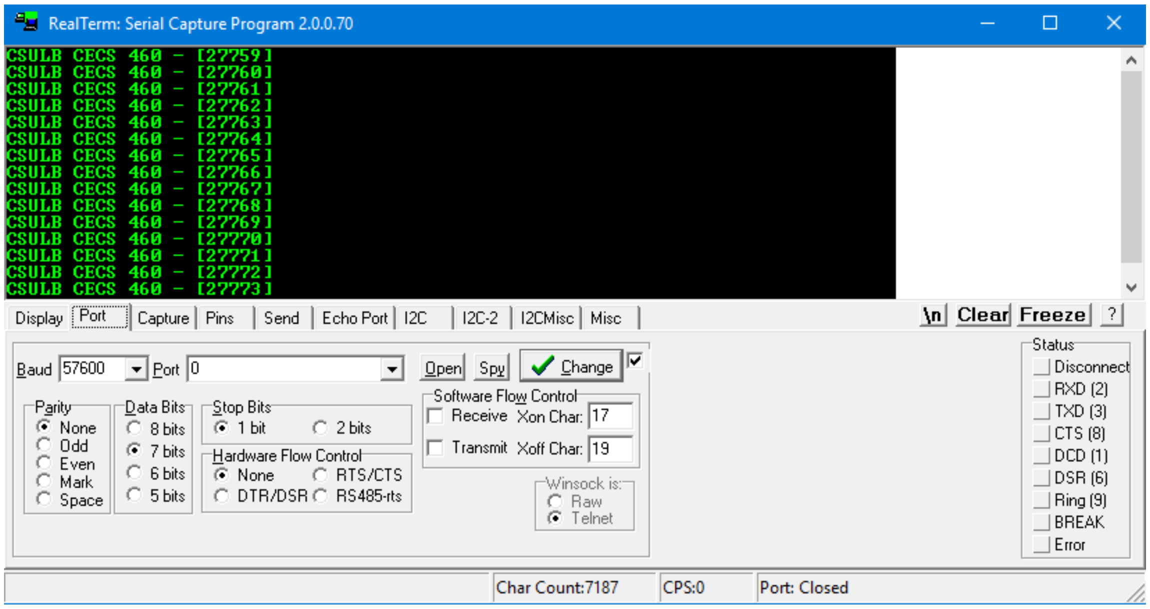
Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Status: Connected, RXD (2), TXD (3), 🟩 CTS (8), DCD (1), DSR (6), Ring (9), BREAK, Error

Char Count:213 CPS:0 Port: 0 921600 7N1 Non

Figure 35: Full UART and TSI RealTerm Output



RealTerm: Serial Capture Program 2.0.0.70

```
CSULB CECS 460 - [27759]
CSULB CECS 460 - [27760]
CSULB CECS 460 - [27761]
CSULB CECS 460 - [27762]
CSULB CECS 460 - [27763]
CSULB CECS 460 - [27764]
CSULB CECS 460 - [27765]
CSULB CECS 460 - [27766]
CSULB CECS 460 - [27767]
CSULB CECS 460 - [27768]
CSULB CECS 460 - [27769]
CSULB CECS 460 - [27770]
CSULB CECS 460 - [27771]
CSULB CECS 460 - [27772]
CSULB CECS 460 - [27773]
```

Baud 57600 Port 0

Parity: [✓ None] Odd Even Mark Space

Data Bits: 8 bits [✓ 7 bits] 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Status: Disconnect, RXD (2), TXD (3), CTS (8), DCD (1), DSR (6), Ring (9), BREAK, Error

Char Count:7187 CPS:0 Port: Closed

Figure 36: Transmit Engine RealTerm Output

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

8. Software Code

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary                 //
// to the CSULB student that created the                          //
// file - any reuse without adequate approval and                 //
// documentation is prohibited                                    //
//                                                                //
// Class:       <CECS 460 SOC>                                    //
// Project:     <Final Project>                                   //
// File name:   Top_Level.v                                       //
//                                                                //
// Created by <Jose Sotelo> on <April 16, 2018>                   //
//                                                                //
// 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 the event other code source are utilized I will             //
// document which portion of code and who is the author           //
//                                                                //
// In submitting this code I acknowledge that plagiarism          //
// in student project work is subject to dismissal from           //
// the class                                                      //
//****************************************************************//
module Top_Level(clk, rst, baud, eight, p_en, ohel, rx, tx, leds);

     input                    clk, rst;
     input      [3:0]         baud;
     input                    eight;
     input                    p_en;
     input                    ohel;
     input                    rx;

     output                   tx;
     output     [15:0]        leds;

     wire                     w_clk, w_rst;
     wire       [3:0]         w_baud;
     wire                     w_eight;
     wire                     w_p_en;
     wire                     w_ohel;
     wire                     w_rx;
     wire       [15:0]        w_leds;


     //**********************************
     //    Core Logic
     //**********************************
     Core_Logic
          uart_core(
               .clk(w_clk),
               .rst(w_rst),
                .baud(w_baud),
                .eight(w_eight),
                .p_en(w_p_en),
                .ohel(w_ohel),
                .rx(w_rx),
                .tx(w_tx),
                .leds(w_leds)
        );

    //**********************************
    //   TSI
    //**********************************
    TSI
        tsi_buf(
                .i_clk(clk),
                .i_rst(rst),
                .i_baud(baud),
                .i_eight(eight),
                .i_p_en(p_en),
                .i_ohel(ohel),
                .i_rx(rx),
                .i_tx(w_tx),
                .i_leds(w_leds),
                .o_clk(w_clk),
                .o_rst(w_rst),
                .o_baud(w_baud),
                .o_eight(w_eight),
                .o_p_en(w_p_en),
                .o_ohel(w_ohel),
                .o_rx(w_rx),
                .o_tx(tx),
                .o_leds(leds)
        );

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Final Project>                           //
// File name:  Core_Logic.v                              //
//                                                       //
// Created by <Jose Sotelo> on <April 16, 2018>          //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//****************************************************************//
module Core_Logic(clk, rst, baud, eight, p_en, ohel, rx, tx, leds);

     input               clk, rst;
     input     [3:0]     baud;
     input               eight, p_en, ohel;
     input               rx;

     output              tx;
     output    [15:0]    leds;

     reg       [15:0]    leds;

     wire                w_rst_s;
     wire                w_write_strobe;
     wire                w_read_strobe;
     wire                w_int_ack;
     wire      [7:0]     w_uart_ds;
     wire                w_uart_int;
     wire                w_interrupt;
     wire      [15:0]    w_port_id;
     wire      [15:0]    w_out_port;

     reg       [15:0]    w_write;
     reg       [15:0]    w_read;

     //************************************
     //   AISO
     //************************************
     Aiso
          aiso(
               .clk(clk),
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
          .rst(rst),
          .rst_s(w_rst_s)
     );

//***********************************
//   Address Decode
//***********************************
always @(*)
begin
     w_write = 16'b0;
     w_read  = 16'b0;

     w_write[w_port_id] = w_write_strobe;
     w_read[w_port_id]  = w_read_strobe;
end

//***********************************
//   UART
//***********************************
UART_Top
     uart(
          .clk(clk),
          .rst(w_rst_s),
          .write(w_write[0]),
          .read(w_read[1:0]),
          .out_port(w_out_port[7:0]),
          .rx(rx),
          .baud(baud),
          .eight(eight),
          .p_en(p_en),
          .ohel(ohel),
          .tx(tx),
          .UART_DS(w_uart_ds),
          .UART_INT(w_uart_int)
     );

//***********************************
//   SR Flop Interrupt
//***********************************
SR_Flop
     sr_interrupt(
          .clk(clk),
          .rst(w_rst_s),
          .S(w_uart_int),
          .R(w_int_ack),
          .Q(w_interrupt)
     );

//***********************************
//   TramelBlaze
//***********************************
tramelblaze_top
     tb_top(
          .CLK(clk),
          .RESET(w_rst_s),
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
                .IN_PORT({8'b0, w_uart_ds}),
                .INTERRUPT(w_interrupt),
                .OUT_PORT(w_out_port),
                .PORT_ID(w_port_id),
                .READ_STROBE(w_read_strobe),
                .WRITE_STROBE(w_write_strobe),
                .INTERRUPT_ACK(w_int_ack)
           );

      //**********************************
      //    Walking LEDs
      //**********************************
      always @(posedge clk, posedge w_rst_s)
      begin
           if(w_rst_s)
                leds <= 16'b0;
           else if(w_write[2])
                leds <= w_out_port;
           else
                leds <= leds;
      end

endmodule
```

```
`timescale 1ns / 1ps
//*******************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Final Project>                           //
// File name:  TSI.v                                     //
//                                                       //
// Created by <Jose Sotelo> on <April 16, 2018>          //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*******************************************************//
module TSI(i_clk, i_rst, i_baud, i_eight, i_p_en, i_ohel, i_rx, i_tx, i_leds,
           o_clk, o_rst, o_baud, o_eight, o_p_en, o_ohel, o_rx, o_tx,
o_leds);


     input                i_clk;
     input                i_rst;
     input      [3:0]     i_baud;
     input                i_eight;
     input                i_p_en;
     input                i_ohel;
     input                i_rx;

     input                i_tx;
     input      [15:0]    i_leds;

     output               o_clk;
     output               o_rst;
     output     [3:0]     o_baud;
     output               o_eight;
     output               o_p_en;
     output               o_ohel;
     output               o_rx;

     output               o_tx;
     output     [15:0]    o_leds;

     // BUFG: Global Clock Simple Buffer
     // 7 Series
     // Xilinx HDL Libraries Guide, version 2012.2
     BUFG
     BUFG_inst (
               .O(o_clk), // 1-bit output: Clock output
               .I(i_clk)  // 1-bit input: Clock input
     );

// IOBUF: Single-ended Bi-directional Buffer
// All devices
// Xilinx HDL Libraries Guide, version 2012.2
IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) rst(
          .O(o_rst), // Buffer output
          .I(i_rst)  // Buffer input
     );

IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) baud[3:0](
          .O(o_baud), // Buffer output
          .I(i_baud)  // Buffer input
     );

IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) eight(
          .O(o_eight), // Buffer output
          .I(i_eight)  // Buffer input
     );

IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) p_en(
          .O(o_p_en), // Buffer output
          .I(i_p_en)  // Buffer input
     );

IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) ohel(
          .O(o_ohel), // Buffer output
          .I(i_ohel)  // Buffer input
     );

IBUF #(
     .IBUF_LOW_PWR("TRUE"), // Low Power - "TRUE", High Perfor = "FALSE"
     .IOSTANDARD("DEFAULT") // Specify the I/O standard
    ) rx(
          .O(o_rx),  // Buffer output
          .I(i_rx)   // Buffer input
     );
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
	// OBUF: Single-ended Output Buffer
	// 7 Series
	// Xilinx HDL Libraries Guide, version 2012.2

	OBUF #(
		.DRIVE(12),                // Specify the output drive strength
		.IOSTANDARD("DEFAULT"), // Specify the output I/O standard
		.SLEW("SLOW")              // Specify the output slew rate
	)tx (
			.O(o_tx),  // Buffer output (connect directly to top-level
port)
			.I(i_tx)   // Buffer input
		);

	OBUF #(
		.DRIVE(12),                // Specify the output drive strength
		.IOSTANDARD("DEFAULT"), // Specify the output I/O standard
		.SLEW("SLOW")              // Specify the output slew rate
	)leds[15:0] (
			.O(o_leds),  // Buffer output (connect directly to top-level
port)
			.I(i_leds)   // Buffer input
		);
endmodule
```

```
`timescale 1ns / 1ps
```

```
//****************************************************//
// This document contains information proprietary     //
// to the CSULB student that created the              //
// file - any reuse without adequate approval and     //
// documentation is prohibited                        //
//                                                    //
// Class:      <CECS 460 SOC>                         //
// Project:    <Project 2>                            //
// File name:  Aiso.v                                 //
//                                                    //
// Created by <Jose Sotelo> on <>                     //
//                                                    //
// 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 the event other code source are utilized I will //
// document which portion of code and who is the author //
//                                                    //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from //
// the class                                          //
//****************************************************//
module Aiso(clk, rst, rst_s);

     input     clk;
     input     rst;

     output    rst_s;

     reg       q1, q2;

     always @(posedge clk, posedge rst)
          if(rst)
               {q1,q2} <= 2'b0;
          else
               {q1,q2} <= {1'b1, q1};

     //*********************************
     //   Assign Statement
     //   Modeling combinational logic
     //*********************************
     assign rst_s = ~q2;

endmodule
```

```
`timescale 1ns / 1ps
```

```
//*****************************************************//
// This document contains information proprietary      //
// to the CSULB student that created the               //
// file - any reuse without adequate approval and      //
// documentation is prohibited                         //
//                                                     //
// Class:      <CECS 460 SOC>                          //
// Project:    <Project 3>                             //
// File name:  UART_Top.v                              //
//                                                     //
// Created by <Jose Sotelo> on <>                      //
//                                                     //
// 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 the event other code source are utilized I will  //
// document which portion of code and who is the author //
//                                                     //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from //
// the class                                           //
//*****************************************************//
module UART_Top(clk, rst, write, read, out_port, rx, baud,
                eight, p_en, ohel, tx, UART_DS, UART_INT);

     input              clk, rst;
     input              write;
     input     [1:0]    read;
     input     [7:0]    out_port;
     input     [3:0]    baud;
     input              eight, p_en, ohel;
     input              rx;

     output             tx;
     output    [7:0]    UART_DS;
     output             UART_INT;

     wire               w_tx_rdy;
     wire               w_rx_rdy;
     wire               w_ped_tx;
     wire               w_ped_rx;
     wire               w_perr;
     wire               w_ferr;
     wire               w_ovf;
     wire      [18:0]   k;
     wire      [7:0]    w_data_to_TB;
     wire      [7:0]    w_status;

     //************************************
     //   Baud
     //************************************
     Baud_Dec
          baudDec(
               .baud(baud),
               .baud_count(k)
          );

     //************************************
     //   Tx Engine
     //************************************
     Tx_Engine
          transmit_engine(
               .clk(clk),
               .rst(rst),
               .load(write),
               .baud_count(k),
               .data_in(out_port),
               .eight(eight),
               .p_en(p_en),
               .ohel(ohel),
               .txrdy(w_tx_rdy),
               .tx(tx)
          );

     //************************************
     //   Rx Engine
     //************************************
     Rx_Engine
          receive_engine(
               .clk(clk),
               .rst(rst),
               .rx(rx),
               .k(k),
               .eight(eight),
               .p_en(p_en),
               .ohel(ohel),
               .read(read[0]),
               .rx_rdy(w_rx_rdy),
               .parity_err(w_perr),
               .frame_err(w_ferr),
               .overflow(w_ovf),
               .data_to_TB(w_data_to_TB)
          );

     assign w_status = {3'b0, w_ovf, w_ferr, w_perr, w_tx_rdy, w_rx_rdy};

     assign UART_DS = (read[1]) ? w_status     :
                      (read[0]) ? w_data_to_TB :
                                  8'b0;

     //************************************
     //   PED Tx
     //   PED Rx
     //************************************
     PED
          tx_ped(
               .clk(clk),
               .rst(rst),
               .ped_in(w_tx_rdy),
     			.ped_out(w_ped_tx)
     		),

     		rx_ped(
     			.clk(clk),
     			.rst(rst),
     			.ped_in(w_rx_rdy),
     			.ped_out(w_ped_rx)
     		);

     	assign UART_INT = w_ped_tx | w_ped_rx;  // wire to SR Flop
     endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//*******************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:       <CECS 460 SOC>                           //
// Project:     <Project 2>                              //
// File name:   SR_Flop.v                                //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*******************************************************//
module SR_Flop(clk, rst, S, R, Q);

     input      clk, rst;
     input      S, R;

     output     Q;

     reg        Q;

     always @(posedge clk, posedge rst)
          if(rst)
               Q <= 1'b0;
          else if(S)
               Q <= 1'b1;
          else if(R)
               Q <= 1'b0;
          else
               Q <= Q;
endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//*******************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 2>                               //
// File name:  Baud_Dec.v                                //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*******************************************************//
module Baud_Dec(baud, baud_count);

     input      [3:0]     baud;
     output     [18:0]    baud_count;

     reg        [18:0]    baud_count;

     // Baud Rate
     // Calculation (1/baud rate) / (1/100MHz) - 1
     always @(*)
          case(baud)
               4'b0000 : baud_count <= 333333 - 1;    // Baud rate 300
               4'b0001 : baud_count <= 83333  - 1;    // Baud rate 1200
               4'b0010 : baud_count <= 41667  - 1;    // Baud rate 2400
               4'b0011 : baud_count <= 20833  - 1;    // Baud rate 4800
               4'b0100 : baud_count <= 10417  - 1;    // Baud rate 9600
               4'b0101 : baud_count <= 5208   - 1;    // Baud rate 19200
               4'b0110 : baud_count <= 2604   - 1;    // Baud rate 38400
               4'b0111 : baud_count <= 1736   - 1;    // Baud rate 57600
               4'b1000 : baud_count <= 868    - 1;    // Baud rate 115200
               4'b1001 : baud_count <= 434    - 1;    // Baud rate 230400
               4'b1010 : baud_count <= 217    - 1;    // Baud rate 460800
               4'b1011 : baud_count <= 109    - 1;    // Baud rate 921600
               default : baud_count <= 333333 - 1;    // Default rate 300
          endcase
endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************//
// This document contains information proprietary      //
// to the CSULB student that created the               //
// file - any reuse without adequate approval and      //
// documentation is prohibited                         //
//                                                     //
// Class:      <CECS 460 SOC>                          //
// Project:    <Project 3>                             //
// File name:  Tx_Engine.v                             //
//                                                     //
// Created by <Jose Sotelo> on <>                      //
//                                                     //
// 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 the event other code source are utilized I will  //
// document which portion of code and who is the author //
//                                                     //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from //
// the class                                           //
//****************************************************//
module Tx_Engine(clk, rst, load, baud_count, data_in,
                 eight, p_en, ohel, txrdy, tx);

     input                clk, rst, load;
     input      [18:0]    baud_count;
     input      [7:0]     data_in;
     input                eight, p_en, ohel;

     output               txrdy;
     output               tx;

     reg                  w_load_d1;

     wire                 w_done;
     wire                 w_do_it;
     wire                 w_btu;
     wire       [7:0]     w_load_data;
     wire                 w_bit10, w_bit9;

     //************************************
     //   SR Flip Flop TxRdy
     //************************************
     SR_Flop_Txrdy
          txrdy_sr_flop(
               .clk(clk),
               .rst(rst),
               .S(w_done),
               .R(load),
               .Q(txrdy)
          );
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
//***********************************
//   SR Flip Flop DoIt
//***********************************
Tx_SR_Flop
     doit_sr_flop(
          .clk(clk),
          .rst(rst),
          .S(w_load_d1),
          .R(w_done),
          .Q(w_do_it)
     );

//***********************************
// 8-bit Loadable Register
//***********************************
Tx_Load_Reg
     tx_ld_reg(
          .clk(clk),
          .rst(rst),
          .ld(load),
          .D(data_in),
          .Q(w_load_data)
     );

//***********************************
// Parity Decoder
//***********************************
Tx_Parity_Gen_Dec
     tx_p_gen(
          .load_data(w_load_data),
          .eight(eight),
          .p_en(p_en),
          .ohel(ohel),
          .bit10(w_bit10),
          .bit9(w_bit9)
     );

//***********************************
//   Register
//***********************************
always @(posedge clk, posedge rst)
     if(rst)
          w_load_d1 <= 1'b0;
     else
          w_load_d1 <= load;

//***********************************
//   Shift Register
//***********************************
Tx_shift_register
     tx_shift_reg(
          .clk(clk),
          .rst(rst),
          .data_in({w_bit10, w_bit9, w_load_data[6:0], 1'b0, 1'b1}),
          .load(w_load_d1),
                .shift(w_btu),
                .sdo(tx)
           );

      //***********************************
      //   Bit Time Counter
      //***********************************
      Tx_bit_time_counter
           tx_btc(
                .clk(clk),
                .rst(rst),
                .baud_count(baud_count),
                .sel({w_do_it, w_btu}),
                .btu(w_btu)
           );

      //***********************************
      //   Bit Counter
      //***********************************
      Tx_bit_counter
           tx_bc(
                .clk(clk),
                .rst(rst),
                .sel({w_do_it, w_btu}),
                .done(w_done)
           );
endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                  //
// file - any reuse without adequate approval and         //
// documentation is prohibited                            //
//                                                        //
// Class:       <CECS 460 SOC>                            //
// Project:     <Project 3>                               //
// File name:   Rx_Engine.v                               //
//                                                        //
// Created by <Jose Sotelo> on <>                         //
//                                                        //
// 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 the event other code source are utilized I will     //
// document which portion of code and who is the author   //
//                                                        //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                              //
//****************************************************************//
module Rx_Engine(clk, rst, rx, k, eight, p_en, ohel, read,
                 rx_rdy, parity_err, frame_err, overflow, data_to_TB);

     input                 clk, rst;
     input                 rx;
     input       [18:0]    k;
     input                 eight, p_en, ohel;
     input                 read;

     output                rx_rdy;
     output                parity_err;
     output                frame_err;
     output                overflow;
     output      [7:0]     data_to_TB;

     wire                  w_start;
     wire                  w_btu;
     wire                  w_done;

     //***********************************
     //    Rx Engine Control
     //***********************************
     Rx_Engine_Control
          control_rx(
               .clk(clk),
               .rst(rst),
               .rx(rx),
               .k(k),
               .eight(eight),
               .p_en(p_en),
               .start(w_start),
                .btu(w_btu),
                .done(w_done)
            );

        //************************************
        //    Rx Engine Datapath
        //************************************
        Rx_DataPath
            datapath_rx(
                .clk(clk),
                .rst(rst),
                .btu(w_btu),
                .start(w_start),
                .rx(rx),
                .eight(eight),
                .p_en(p_en),
                .even(~ohel),
                .read(read),
                .done(w_done),
                .rx_rdy(rx_rdy),
                .parity_err(parity_err),
                .frame_err(frame_err),
                .overflow(overflow),
                .data_to_TB(data_to_TB)
            );
    endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//*******************************************************//
// This document contains information proprietary       //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:       <CECS 460 SOC>                           //
// Project:     <Project 2>                              //
// File name:   PED.v                                    //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*******************************************************//
module PED(clk, rst, ped_in, ped_out);

     input      clk, rst;
     input      ped_in;

     output     ped_out;

     reg        q1, q2;

     always @(posedge clk, posedge rst)
          if(rst)
               {q1, q2} <= 2'b0;
          else
               {q1,q2} <= {ped_in, q1};

     //************************************
     //   Assign Statement
     //   Modeling combinational logic
     //************************************
     assign ped_out = ~q2 & q1;

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary              //
// to the CSULB student that created the                       //
// file - any reuse without adequate approval and              //
// documentation is prohibited                                 //
//                                                             //
// Class:       <CECS 460 SOC>                                 //
// Project:     <Project 2>                                    //
// File name:   SR_Flop_Txrdy.v                                //
//                                                             //
// Created by <Jose Sotelo> on <>                              //
//                                                             //
// 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 the event other code source are utilized I will          //
// document which portion of code and who is the author        //
//                                                             //
// In submitting this code I acknowledge that plagiarism       //
// in student project work is subject to dismissal from        //
// the class                                                   //
//****************************************************************//
module SR_Flop_Txrdy(clk, rst, S, R, Q);

     input     clk, rst;
     input     S, R;

     output    Q;

     reg       Q;

     always @(posedge clk, posedge rst)
          if(rst)
               Q <= 1'b1;
          else if(S)
               Q <= 1'b1;
          else if(R)
               Q <= 1'b0;
          else
               Q <= Q;

endmodule
```

Prepared by:	Date:	Revision:
Jose Sotelo	May 14, 2018	2.1

```
`timescale 1ns / 1ps
//*******************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 2>                               //
// File name:  Tx_Load_Reg.v                             //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*******************************************************//
module Tx_Load_Reg(clk, rst, ld, D, Q);

     input                 clk, rst, ld;
     input      [7:0]      D;

     output     [7:0]      Q;

     reg        [7:0]      Q;

     always @(posedge clk, posedge rst)
          if(rst)
          begin
               Q <= 8'b0;
          end
          else if(ld)
          begin
               Q <= D;
          end
          else
          begin
               Q <= Q;
          end
endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//*****************************************************//
// This document contains information proprietary       //
// to the CSULB student that created the                //
// file - any reuse without adequate approval and       //
// documentation is prohibited                          //
//                                                      //
// Class:       <CECS 460 SOC>                          //
// Project:     <Project 2>                             //
// File name:   Tx_Parity_Gen_Dec.v                     //
//                                                      //
// Created by <Jose Sotelo> on <>                       //
//                                                      //
// 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 the event other code source are utilized I will   //
// document which portion of code and who is the author //
//                                                      //
//*****************************************************//
module Tx_Parity_Gen_Dec(load_data, eight, p_en, ohel, bit10, bit9);

     input     [7:0]     load_data;
     input               eight, p_en, ohel;

     output              bit10, bit9;

     reg                 EP, OP;
     reg                 bit10, bit9;

    //*********************************
     //   y = f(x)
     //   Function is used to determine
     //   Odd/Even parity bits
     //*********************************
      always @ (*)
         begin
          EP   <= (eight) ?   ^load_data[7:0]    :   ^load_data[6:0];
          OP   <= (eight) ? ~(^load_data[7:0])   : ~(^load_data[6:0]);
         end

     always @(*)
          case({eight, p_en, ohel})
               3'b000 : {bit10, bit9}  <=  2'b11;                // 7N1
               3'b001 : {bit10, bit9}  <=  2'b11;                // 7N1
               3'b010 : {bit10, bit9}  <= {1'b1, EP};            // 7E1
               3'b011 : {bit10, bit9}  <= {1'b1, OP};            // 7O1
               3'b100 : {bit10, bit9}  <= {1'b1, load_data[7]};  // 8N1
               3'b101 : {bit10, bit9}  <= {1'b1, load_data[7]};  // 8N1
               3'b110 : {bit10, bit9}  <= {EP,   load_data[7]};  // 8E1
               3'b111 : {bit10, bit9}  <= {OP,   load_data[7]};  // 8O1
          endcase

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary         //
// to the CSULB student that created the                  //
// file - any reuse without adequate approval and         //
// documentation is prohibited                            //
//                                                        //
// Class:       <CECS 460 SOC>                            //
// Project:     <Project 2>                               //
// File name:   Tx_shift_register .v                      //
//                                                        //
// Created by <Jose Sotelo> on <>                         //
//                                                        //
// 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 the event other code source are utilized I will     //
// document which portion of code and who is the author   //
//                                                        //
// In submitting this code I acknowledge that plagiarism  //
// in student project work is subject to dismissal from   //
// the class                                              //
//****************************************************************//
module Tx_shift_register(clk, rst, data_in, load, shift, sdo);

     input                clk, rst;
     input      [10:0]    data_in;
     input                load;
     input                shift;

     output               sdo;

     reg        [10:0]    shift_reg;

     always @(posedge clk, posedge rst)
          if(rst)
               shift_reg <= 11'b11111_111111;
          else if(load)
               shift_reg <= data_in;
          else if(shift)
               shift_reg <= {1'b1, shift_reg[10:1]};
          else
               shift_reg <= shift_reg;

     assign sdo = shift_reg[0];

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 2>                               //
// File name:  Tx_bit_time_counter.v                     //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
//****************************************************************//
module Tx_bit_time_counter(clk, rst, baud_count, sel, btu);

     input                clk, rst;
     input     [18:0]     baud_count;
     input     [1:0]      sel;      // concat 1-bit do_it & btu wires

     output               btu;

     reg       [18:0]     mux_out;
     reg       [18:0]     bit_time_count;

     // MUX Selector
     always @(*)
          case(sel)
               2'b00      :     mux_out <= 19'b0;
               2'b01      :     mux_out <= 19'b0;
               2'b10      :     mux_out <= bit_time_count + 19'b1;
               2'b11      :     mux_out <= 19'b0;
               default    :     mux_out <= 19'b0;
          endcase

     // 19 bit Register
     always @(posedge clk, posedge rst)
          if(rst)
               bit_time_count <= 19'b0;
          else
               bit_time_count <= mux_out;

     // Comparator
     assign btu = (bit_time_count == baud_count) ? 1'b1 : 1'b0;

endmodule
```

```
`timescale 1ns / 1ps
//********************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                        //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 2>                               //
// File name:  Tx_bit_counter.v                          //
//                                                        //
// Created by <Jose Sotelo> on <>                        //
//                                                        //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                        //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//********************************************************//
module Tx_bit_counter(clk, rst, sel, done);

     input                clk, rst;
     input      [1:0]     sel;

     output               done;

     reg        [3:0]     mux_out;
     reg        [3:0]     bit_count;


     // MUX Select
     always @(*)
          case(sel)
               2'b00 : mux_out <= 4'b0;
               2'b01 : mux_out <= 4'b0;
               2'b10 : mux_out <= bit_count;
               2'b11 : mux_out <= bit_count + 4'b1;
          endcase

     // 4-bit Register
     always @(posedge clk, posedge rst)
          if(rst)
               bit_count <= 4'b0;
          else
               bit_count <= mux_out;

     assign done = (bit_count == 11) ? 1'b1 : 1'b0;

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//*****************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:       <CECS 460 SOC>                           //
// Project:     <Project 3>                              //
// File name:   Rx_Engine_Control.v                      //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//*****************************************************//
module Rx_Engine_Control(clk, rst, rx, k, eight, p_en,
                         start, btu, done);

     input                clk, rst;
     input                rx;
     input        [18:0]  k;
     input                eight;
     input                p_en;

     output               start;
     output               btu;
     output               done;

      wire                w_do_it;

     //**********************************
     //   Rx FSM
     //**********************************
     Rx_FSM
          rx_fsm(
               .clk(clk),
               .rst(rst),
               .rx(rx),
               .btu(btu),
               .done(done),
               .start(start),
               .do_it(w_do_it)
          );
```

```
    //***********************************
    //   Rx Bit Time Counter
    //***********************************
     Rx_bit_time_counter
         rx_btc(
              .clk(clk),
              .rst(rst),
              .start(start),
              .k(k),
              .sel({w_do_it, btu}),
              .btu(btu)
         );

    //***********************************
    //   Rx Bit Counter
    //***********************************
    Rx_bit_counter
         rx_bc(
              .clk(clk),
              .rst(rst),
              .sel({w_do_it, btu}),
              .eight(eight),
              .p_en(p_en),
              .done(done)
         );
endmodule
```

```
`timescale 1ns / 1ps
//*****************************************************//
// This document contains information proprietary       //
// to the CSULB student that created the                //
// file - any reuse without adequate approval and       //
// documentation is prohibited                          //
//                                                      //
// Class:      <CECS 460 SOC>                           //
// Project:    <Project 3>                              //
// File name:  Rx_FSM.v                                 //
//                                                      //
// Created by <Jose Sotelo> on <>                       //
//                                                      //
// 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 the event other code source are utilized I will   //
// document which portion of code and who is the author //
//                                                      //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from //
// the class                                            //
//*****************************************************//
module Rx_FSM(clk, rst, rx, btu, done, start, do_it);

     input          clk, rst;
     input          rx, btu, done;

     output         start, do_it;

     reg            start, do_it;

     reg  [1:0]     CurrentState, NxtState;
     reg            NxtStart, NxtDo_It;

     parameter      s0 = 2'b00,
                    s1 = 2'b01,
                    s2 = 2'b10;

     always @(posedge clk, posedge rst)
     begin
          if(rst)
               {CurrentState, start, do_it} <= {s0, 2'b00};
          else
               {CurrentState, start, do_it} <= {NxtState, NxtStart,
NxtDo_It};
     end

     always @(*)
     begin
          NxtStart = 0;
          NxtDo_It = 0;
          NxtState = CurrentState;
          case(CurrentState)
               s0:
               begin
                    {NxtState, NxtStart, NxtDo_It} = (rx) ? {s0, 2'b00} :
                                                            {s1, 2'b11};
               end
               s1:
               begin
                    {NxtState, NxtStart, NxtDo_It} =
                                             (rx)           ? {s0, 2'b00} :
                                             (~rx && ~btu)  ? {s1, 2'b11} :
                                                              {s2, 2'b01};
               end
               s2:
               begin
                    {NxtState, NxtStart, NxtDo_It} = (~done) ? {s2, 2'b01} :
                                                               {s0, 2'b00};
               end
               default   : {NxtState, NxtStart, NxtDo_It} = {s0, 2'b00};
          endcase
     end
endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************//
// This document contains information proprietary      //
// to the CSULB student that created the               //
// file - any reuse without adequate approval and      //
// documentation is prohibited                         //
//                                                     //
// Class:      <CECS 460 SOC>                          //
// Project:    <Project 3>                             //
// File name:  Rx_bit_time_counter.v                   //
//                                                     //
// Created by <Jose Sotelo> on <>                      //
//                                                     //
// 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 the event other code source are utilized I will  //
// document which portion of code and who is the author //
//****************************************************//
module Rx_bit_time_counter(clk, rst, start, k, sel, btu);

     input                clk, rst;
     input                start;
     input      [18:0]    k;
     input      [1:0]     sel;       // concat 1-bit do_it & btu wires

     output               btu;

     reg        [18:0]    mux_out;
     reg        [18:0]    bit_time_count;

     wire       [18:0]    mux_baud_count_out;

     // MUX Selector
     always @(*)
          case(sel)
               2'b00     :    mux_out <= 19'b0;
               2'b01     :    mux_out <= 19'b0;
               2'b10     :    mux_out <= bit_time_count + 19'b1;
               2'b11     :    mux_out <= 19'b0;
               default   :    mux_out <= 19'b0;
          endcase

     // 19 bit Register
     always @(posedge clk, posedge rst)
          if(rst)
               bit_time_count <= 19'b0;
          else
               bit_time_count <= mux_out;
     // Comparator
     assign btu = (bit_time_count == mux_baud_count_out) ? 1'b1 : 1'b0;

     // Buad Mux Select
     assign mux_baud_count_out = (start) ? (k >> 1) : (k);
endmodule
```

```
`timescale 1ns / 1ps
//*****************************************************//
// This document contains information proprietary       //
// to the CSULB student that created the                //
// file - any reuse without adequate approval and       //
// documentation is prohibited                          //
//                                                      //
// Class:       <CECS 460 SOC>                          //
// Project:     <Project 3>                             //
// File name:   Rx_bit_counter.v                        //
//                                                      //
// Created by <Jose Sotelo> on <>                       //
//                                                      //
// 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 the event other code source are utilized I will   //
// document which portion of code and who is the author //
//                                                      //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                            //
//*****************************************************//
module Rx_bit_counter(clk, rst, sel, eight, p_en, done);

     input                clk, rst;
     input      [1:0]     sel;
     input                eight, p_en;

     output               done;

     reg        [3:0]     mux_out;
     reg        [3:0]     bit_count;
     reg        [3:0]     num_bits;


     // MUX Select
     always @(*)
     begin
          case(sel)
               2'b00     :    mux_out <= 4'b0;
               2'b01     :    mux_out <= 4'b0;
               2'b10     :    mux_out <= bit_count;
               2'b11     :    mux_out <= bit_count + 4'b1;
               default   :    mux_out <= 4'b0;
          endcase
     end

     // 4-bit Register
     always @(posedge clk, posedge rst)
          if(rst)
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
                bit_count <= 4'b0;
            else
                bit_count <= mux_out;

        // f(x) used to determine the bits
        always @(*)
        begin
            case({eight, p_en})
                2'b00      :    num_bits = 4'h9; // 7N1
                2'b01      :    num_bits = 4'hA; // 7E1/7O1
                2'b10      :    num_bits = 4'hA; // 8N1
                2'b11      :    num_bits = 4'hB; // 8O1/8E1
                default    :    num_bits = 4'hA; // 7N1
            endcase
        end

        // Comparator
        assign done = (bit_count == num_bits) ? 1'b1 : 1'b0;

endmodule
```

```
`timescale 1ns / 1ps
//********************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                        //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 2>                               //
// File name:  Rx_DataPath.v                             //
//                                                        //
// Created by <Jose Sotelo> on <>                        //
//                                                        //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                        //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//********************************************************//
module Rx_DataPath(clk, rst, btu, start, rx,
                   eight, p_en, even, done, read,
                   rx_rdy, parity_err, frame_err, overflow, data_to_TB);

     input                clk, rst;
     input                btu, start, rx;
     input                eight, p_en;
     input                even, done;
     input                read;

     output               rx_rdy;
     output               parity_err;
     output               frame_err;
     output               overflow;
     output     [7:0]     data_to_TB;

     reg                  stop_bit_mux_out;

     wire       [9:0]     w_shift_reg_out;
     wire       [9:0]     w_remap;
     wire                 w_p_gen_mux;
     wire                 w_p_gen_even_mux;
     wire                 w_p_bit_mux;
     wire                 w_perr;
     wire                 w_ferr;
     wire                 w_ovf;

     //************************************
     //   Rx Shift Register
     //************************************
     Rx_Shift_Reg
     rx_shift_reg(
          .clk(clk),
          .rst(rst),
          .sh(btu & ~start),
          .sdi(rx),
          .data_out(w_shift_reg_out)
     );

//**********************************
//   Rx ReMap
//**********************************
Rx_Remap
     rx_remap(
          .sel({eight, p_en}),
          .data_in(w_shift_reg_out),
          .remap(w_remap)
     );

assign w_p_gen_mux = (eight) ? w_remap[7] : 1'b0;

assign w_p_gen_even_mux = (even) ? (^{w_p_gen_mux, w_remap[6:0]}) :
                                   ~(^{w_p_gen_mux, w_remap[6:0]});

assign w_p_bit_mux = (eight) ? w_remap[8] : w_remap[7];

assign w_perr = p_en & (^{w_p_gen_even_mux, w_p_bit_mux}) & done;

//**********************************
//   Stop Bit Select MUX
//**********************************
always @(*)
     case({eight, p_en})
          2'b00     :    stop_bit_mux_out <= w_remap[7];
          2'b01     :    stop_bit_mux_out <= w_remap[8];
          2'b10     :    stop_bit_mux_out <= w_remap[8];
          2'b11     :    stop_bit_mux_out <= w_remap[9];
          default   :    stop_bit_mux_out <= w_remap[7];
     endcase

//**********************************
//   SR Rx Ready Flop
//**********************************
Rx_SR_Flop
     rx_ready(
          .clk(clk),
          .rst(rst),
          .S(done),
          .R(read),
          .Q(rx_rdy)
     );

//**********************************
//   SR Rx Parity Error
//**********************************
Rx_SR_Flop
```

```
		p_err(
			.clk(clk),
			.rst(rst),
			.S(w_perr),
			.R(read),
			.Q(parity_err)
		);

	//***********************************
	//	SR Rx Framing Error
	//***********************************
	Rx_SR_Flop
		framing_err(
			.clk(clk),
			.rst(rst),
			.S(done & ~stop_bit_mux_out),
			.R(read),
			.Q(frame_err)
		);

	//***********************************
	//	SR Rx Overflow Error
	//***********************************
	Rx_SR_Flop
		ovf_err(
			.clk(clk),
			.rst(rst),
			.S(done & rx_rdy),
			.R(read),
			.Q(overflow)
		);

	//***********************************
	//	Data sent to TramelBlaze
	//***********************************
	assign data_to_TB = (eight) ? w_remap[7:0] : {1'b0, w_remap[6:0]};
endmodule
```

```
`timescale 1ns / 1ps
```

```
//*****************************************************//
// This document contains information proprietary      //
// to the CSULB student that created the               //
// file - any reuse without adequate approval and      //
// documentation is prohibited                         //
//                                                     //
// Class:       <CECS 460 SOC>                         //
// Project:     <Project 3>                            //
// File name:   Rx_Shift_Reg.v                         //
//                                                     //
// Created by <Jose Sotelo> on <>                      //
//                                                     //
// 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 the event other code source are utilized I will  //
// document which portion of code and who is the author //
//                                                     //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from //
// the class                                           //
//*****************************************************//
module Rx_Shift_Reg(clk, rst, sh, sdi, data_out);

     input               clk, rst;
     input               sh;
     input               sdi;      // Rx input

     output    [9:0]     data_out;

     reg       [9:0]     data_out;

     always @(posedge clk, posedge rst)
          if(rst)
               data_out <= 10'b0;
          else if(sh)
               data_out <= {sdi, data_out[9:1]};
          else
               data_out <= data_out;


endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
`timescale 1ns / 1ps
//****************************************************************//
// This document contains information proprietary        //
// to the CSULB student that created the                 //
// file - any reuse without adequate approval and        //
// documentation is prohibited                           //
//                                                       //
// Class:      <CECS 460 SOC>                            //
// Project:    <Project 3>                               //
// File name:  Rx_Remap.v                                //
//                                                       //
// Created by <Jose Sotelo> on <>                        //
//                                                       //
// 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 the event other code source are utilized I will    //
// document which portion of code and who is the author  //
//                                                       //
// In submitting this code I acknowledge that plagiarism //
// in student project work is subject to dismissal from  //
// the class                                             //
//****************************************************************//
module Rx_Remap(sel, data_in, remap);

     input     [1:0]     sel;           // eight and p_en
     input     [9:0]     data_in;       // shift reg data

     output    [9:0]     remap;         // remap combo

     reg       [9:0]     remap;

     always @(*)
          case(sel)
               2'b00     :    remap <= {2'b00, data_in[9:2]};
               2'b01     :    remap <= {1'b0,  data_in[9:1]};
               2'b10     :    remap <= {1'b0,  data_in[9:1]};
               2'b11     :    remap <= data_in;
               default   :    remap <= {2'b00, data_in[9:2]};
          endcase

endmodule
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
; Jose Sotelo
; CECS 460
; Full duplex UART and TramelBlaze Assembly

;========================================================
;   Define Variables Alias
;========================================================
ZERO                EQU     0000
ONE                 EQU     0001
PORT                EQU     0001
LED_PORT            EQU     0005
RX_RDY              EQU     0001
TX_RDY              EQU     0002

ASCII_BS            EQU     0008
ASCII_TAB           EQU     0009
ASCII_LF            EQU     000A
ASCII_CR            EQU     000D
ASCII_SLASH         EQU     002F
ASCII_EQUALS        EQU     003D
ASCII_ASTERISK      EQU     002A
ASCII_AT            EQU     0040
ASCII_DOT           EQU     002E
ASCII_SPACE         EQU     0020
ASCII_GREATER       EQU     003E
ASCII_COLON         EQU     003A
ASCII_B_SLASH       EQU     005C

ASCII_ZERO          EQU     0030
ASCII_ONE           EQU     0031
ASCII_TWO           EQU     0032
ASCII_THREE         EQU     0033
ASCII_FOUR          EQU     0034
ASCII_FIVE          EQU     0035
ASCII_SIX           EQU     0036
ASCII_SEVEN         EQU     0037
ASCII_EIGHT         EQU     0038
ASCII_NINE          EQU     0039
ASCII_A             EQU     0041
ASCII_B             EQU     0042
ASCII_C             EQU     0043
ASCII_D             EQU     0044
ASCII_E             EQU     0045
ASCII_F             EQU     0046
ASCII_G             EQU     0047
ASCII_H             EQU     0048
ASCII_I             EQU     0049
ASCII_J             EQU     004A
ASCII_K             EQU     004B
ASCII_L             EQU     004C
ASCII_M             EQU     004D
ASCII_N             EQU     004E
ASCII_O             EQU     004F
ASCII_P             EQU     0050
ASCII_Q             EQU     0051
```

```
ASCII_R              EQU     0052
ASCII_S              EQU     0053
ASCII_T              EQU     0054
ASCII_U              EQU     0055
ASCII_V              EQU     0056
ASCII_W              EQU     0057
ASCII_X              EQU     0058
ASCII_Y              EQU     0059
ASCII_Z              EQU     005A

;========================================================
;   Register Alias
;========================================================
TEMP_REG             EQU     R0
POINTER              EQU     R1
COUNTER              EQU     R2
COUNT                EQU     R3
UART_STATUS          EQU     R4
CURRENT_CASE         EQU     R5
CHAR_COUNT           EQU     R6
STORE_DATA           EQU     R7

;========================================================
;   Start
;   Startup Code Initialization
;   only executed at startup (reset)
;========================================================
START
        LOAD    TEMP_REG,     ZERO
        LOAD    POINTER,      ZERO
        LOAD    COUNTER,      ZERO
        LOAD    COUNT,        ZERO
        LOAD    UART_STATUS,  ZERO
        LOAD    CURRENT_CASE, ONE

        CALL    BANNER
        CALL    PROMPT_USER
        CALL    TOWN
        CALL    BACKSPACE
        CALL    CRLF

        ENINT
        JUMP    MAIN

;========================================================
;   Banner Subroutine
;   Banner is executed on startup
;   /////////////////////////////////
;       CSULB CECS 460
;       Jose Sotelo
;   /////////////////////////////////
;========================================================
BANNER

            LOAD    TEMP_REG,   ASCII_SLASH
```

```
ADD_SLASH   STORE   TEMP_REG,   COUNTER
            ADD     COUNTER,    0001

            COMP    COUNTER,    0028
            JUMPC   ADD_SLASH

            LOAD    TEMP_REG,   ASCII_CR
            STORE   TEMP_REG,   0029

            LOAD    TEMP_REG,   ASCII_LF
            STORE   TEMP_REG,   002A

            LOAD    TEMP_REG,   ASCII_TAB
            STORE   TEMP_REG,   002B

            LOAD    TEMP_REG,   ASCII_C
            STORE   TEMP_REG,   002C

            LOAD    TEMP_REG,   ASCII_S
            STORE   TEMP_REG,   002D

            LOAD    TEMP_REG,   ASCII_U
            STORE   TEMP_REG,   002E

            LOAD    TEMP_REG,   ASCII_L
            STORE   TEMP_REG,   002F

            LOAD    TEMP_REG,   ASCII_B
            STORE   TEMP_REG,   0030

            LOAD    TEMP_REG,   ASCII_SPACE
            STORE   TEMP_REG,   0031

            LOAD    TEMP_REG,   ASCII_C
            STORE   TEMP_REG,   0032

            LOAD    TEMP_REG,   ASCII_E
            STORE   TEMP_REG,   0033

            LOAD    TEMP_REG,   ASCII_C
            STORE   TEMP_REG,   0034

            LOAD    TEMP_REG,   ASCII_S
            STORE   TEMP_REG,   0035

            LOAD    TEMP_REG,   ASCII_SPACE
            STORE   TEMP_REG,   0036

            LOAD    TEMP_REG,   ASCII_FOUR
            STORE   TEMP_REG,   0037

            LOAD    TEMP_REG,   ASCII_SIX
            STORE   TEMP_REG,   0038

            LOAD    TEMP_REG,   ASCII_ZERO
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
STORE   TEMP_REG,   0039

LOAD    TEMP_REG,   ASCII_CR
STORE   TEMP_REG,   003A

LOAD    TEMP_REG,   ASCII_LF
STORE   TEMP_REG,   003B

LOAD    TEMP_REG,   ASCII_TAB
STORE   TEMP_REG,   003C

LOAD    TEMP_REG,   ASCII_SPACE
STORE   TEMP_REG,   003D

LOAD    TEMP_REG,   ASCII_J
STORE   TEMP_REG,   003E

LOAD    TEMP_REG,   ASCII_O
STORE   TEMP_REG,   003F

LOAD    TEMP_REG,   ASCII_S
STORE   TEMP_REG,   0040

LOAD    TEMP_REG,   ASCII_E
STORE   TEMP_REG,   0041

LOAD    TEMP_REG,   ASCII_SPACE
STORE   TEMP_REG,   0042

LOAD    TEMP_REG,   ASCII_S
STORE   TEMP_REG,   0043

LOAD    TEMP_REG,   ASCII_O
STORE   TEMP_REG,   0044

LOAD    TEMP_REG,   ASCII_T
STORE   TEMP_REG,   0045

LOAD    TEMP_REG,   ASCII_E
STORE   TEMP_REG,   0046

LOAD    TEMP_REG,   ASCII_L
STORE   TEMP_REG,   0047

LOAD    TEMP_REG,   ASCII_O
STORE   TEMP_REG,   0048

LOAD    TEMP_REG,   ASCII_CR
STORE   TEMP_REG,   0049

LOAD    TEMP_REG,   ASCII_LF
STORE   TEMP_REG,   004A

LOAD    TEMP_REG,   ASCII_TAB
STORE   TEMP_REG,   004B
```

```
            LOAD     TEMP_REG,    ASCII_SPACE
            STORE    TEMP_REG,    004C

            LOAD     TEMP_REG,    ASCII_SPACE
            STORE    TEMP_REG,    004D

            LOAD     TEMP_REG,    ASCII_SPACE
            STORE    TEMP_REG,    004E

            LOAD     TEMP_REG,    ASCII_SPACE
            STORE    TEMP_REG,    004F

            LOAD     TEMP_REG,    ASCII_U
            STORE    TEMP_REG,    0050

            LOAD     TEMP_REG,    ASCII_A
            STORE    TEMP_REG,    0051

            LOAD     TEMP_REG,    ASCII_R
            STORE    TEMP_REG,    0052

            LOAD     TEMP_REG,    ASCII_T
            STORE    TEMP_REG,    0053

            LOAD     TEMP_REG,    ASCII_CR
            STORE    TEMP_REG,    0054

            LOAD     TEMP_REG,    ASCII_LF
            STORE    TEMP_REG,    0055

            LOAD     TEMP_REG,    ASCII_SLASH
            LOAD     COUNT,       0056
LAST        STORE    TEMP_REG,    COUNT

            ADD      COUNT,       0001

            COMP     COUNT,       007E
            JUMPC    LAST

            LOAD     TEMP_REG,    ASCII_CR
            STORE    TEMP_REG,    007F

            LOAD     TEMP_REG,    ASCII_LF
            STORE    TEMP_REG,    0080


;=======================================================
;   Prompt User Subroutine
;   Allows the user to type
;   C:\User>
;=======================================================
PROMPT_USER

            LOAD     TEMP_REG,    ASCII_C
            STORE   TEMP_REG,   0081

            LOAD    TEMP_REG,   ASCII_COLON
            STORE   TEMP_REG,   0082

            LOAD    TEMP_REG,   ASCII_B_SLASH
            STORE   TEMP_REG,   0083

            LOAD    TEMP_REG,   ASCII_U
            STORE   TEMP_REG,   0084

            LOAD    TEMP_REG,   ASCII_S
            STORE   TEMP_REG,   0085

            LOAD    TEMP_REG,   ASCII_E
            STORE   TEMP_REG,   0086

            LOAD    TEMP_REG,   ASCII_R
            STORE   TEMP_REG,   0087

            LOAD    TEMP_REG,   ASCII_GREATER
            STORE   TEMP_REG,   0088

;========================================================
;   TOWN Subroutine
;   Prints the designer's hometown
;   Hometown: Boyle Heights
;========================================================
TOWN
            LOAD    TEMP_REG,   ASCII_H
            STORE   TEMP_REG,   0089

            LOAD    TEMP_REG,   ASCII_O
            STORE   TEMP_REG,   008A

            LOAD    TEMP_REG,   ASCII_M
            STORE   TEMP_REG,   008B

            LOAD    TEMP_REG,   ASCII_E
            STORE   TEMP_REG,   008C

            LOAD    TEMP_REG,   ASCII_T
            STORE   TEMP_REG,   008D

            LOAD    TEMP_REG,   ASCII_O
            STORE   TEMP_REG,   008E

            LOAD    TEMP_REG,   ASCII_W
            STORE   TEMP_REG,   008F

            LOAD    TEMP_REG,   ASCII_N
            STORE   TEMP_REG,   0090

            LOAD    TEMP_REG,   ASCII_COLON
            STORE   TEMP_REG,   0091
```

```
LOAD     TEMP_REG,    ASCII_SPACE
STORE    TEMP_REG,    0092

LOAD     TEMP_REG,    ASCII_B
STORE    TEMP_REG,    0093

LOAD     TEMP_REG,    ASCII_O
STORE    TEMP_REG,    0094

LOAD     TEMP_REG,    ASCII_Y
STORE    TEMP_REG,    0095

LOAD     TEMP_REG,    ASCII_L
STORE    TEMP_REG,    0096

LOAD     TEMP_REG,    ASCII_E
STORE    TEMP_REG,    0097

LOAD     TEMP_REG,    ASCII_SPACE
STORE    TEMP_REG,    0098

LOAD     TEMP_REG,    ASCII_H
STORE    TEMP_REG,    0099

LOAD     TEMP_REG,    ASCII_E
STORE    TEMP_REG,    009A

LOAD     TEMP_REG,    ASCII_I
STORE    TEMP_REG,    009B

LOAD     TEMP_REG,    ASCII_G
STORE    TEMP_REG,    009C

LOAD     TEMP_REG,    ASCII_H
STORE    TEMP_REG,    009D

LOAD     TEMP_REG,    ASCII_T
STORE    TEMP_REG,    009E

LOAD     TEMP_REG,    ASCII_S
STORE    TEMP_REG,    009F

LOAD     TEMP_REG,    ASCII_CR
STORE    TEMP_REG,    00A0

LOAD     TEMP_REG,    ASCII_LF
STORE    TEMP_REG,    00A1
```

```
;=======================================================
;   Backspace
;   Allows the user to move the display cursor one
;   position backwards and deletes the character at
;   that position
;=======================================================
BACKSPACE
            LOAD    TEMP_REG,   ASCII_BS
            STORE   TEMP_REG,   00A2

            LOAD    TEMP_REG,   ASCII_SPACE
            STORE   TEMP_REG,   00A3

            LOAD    TEMP_REG,   ASCII_BS
            STORE   TEMP_REG,   00A4

            RETURN

;=======================================================
;   Carriage Return and line feed
;   They're used to note the termination of a line
;=======================================================
CRLF
            LOAD    TEMP_REG,   ASCII_CR
            STORE   TEMP_REG,   00A5

            LOAD    TEMP_REG,   ASCII_LF
            STORE   TEMP_REG,   00A6

            RETURN

;=======================================================
;   TX Subroutine
;=======================================================
TX

            COMP    CURRENT_CASE, ZERO
            RETURNZ

            FETCH   TEMP_REG,   POINTER
            OUTPUT  TEMP_REG,   ZERO
            ADD     POINTER,    ONE

            COMP    CURRENT_CASE, 0001
            JUMPZ   SHOW_BANNER

            COMP    CURRENT_CASE, 0002
            JUMPZ   SHOW_PROMPT_USER

            COMP    CURRENT_CASE, 0003
            JUMPZ   SHOW_TOWN

            COMP    CURRENT_CASE, 0004
            JUMPZ   SHOW_BACKSPACE
            COMP    CURRENT_CASE, 0005
            JUMPZ   SHOW_CRLF

            COMP    CURRENT_CASE, 0006
            JUMPZ   SHOW_COUNT

            RETURN

SHOW_BANNER

            COMP    POINTER,      0081
            RETURNC
            LOAD    CURRENT_CASE, 0002
            RETURN


SHOW_PROMPT_USER

            COMP    POINTER,      0089
            RETURNC
            LOAD    CURRENT_CASE, 0000
            RETURN

SHOW_TOWN

            COMP    POINTER,      00A2
            RETURNC
            LOAD    POINTER,      0081
            LOAD    CURRENT_CASE, 0002
            RETURN

SHOW_BACKSPACE

            COMP    POINTER,      00A5
            RETURNC
            LOAD    CURRENT_CASE, 0000
            RETURN

SHOW_CRLF

            COMP    POINTER,      00A7
            RETURNC
            LOAD    POINTER,      0081
            LOAD    CURRENT_CASE, 0002
            RETURN

SHOW_COUNT

            COMP    POINTER,      00A9
            RETURNC
            LOAD    POINTER,      00A5
            LOAD    CURRENT_CASE, 0005
            RETURN
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
;========================================================
;   RX Subroutine
;========================================================
RX

            COMP    CURRENT_CASE,   ZERO
            RETURNNZ

            INPUT   STORE_DATA,     ZERO
            COMP    STORE_DATA,     ZERO
            RETURNZ

            COMP    STORE_DATA,     ASCII_ASTERISK
            JUMPZ   RECEIVE_TOWN

            COMP    STORE_DATA,     ASCII_BS

            COMP    STORE_DATA,     ASCII_CR
            JUMPZ   RECEIVE_CRLF

            COMP    STORE_DATA,     ASCII_AT
            JUMPZ   RECIEVE_AT

            ;used to display char count
            ADD     CHAR_COUNT,     0001
            OUTPUT  STORE_DATA,     ZERO
            COMP    CHAR_COUNT,     0029
            JUMPZ   RECEIVE_CRLF

            RETURN

RECEIVE_TOWN

            LOAD    CURRENT_CASE, 0003
            LOAD    POINTER,      0089
            LOAD    TEMP_REG,     0000
            OUTPUT  TEMP_REG,     0000
            LOAD    CHAR_COUNT,   0000
            RETURN

RECEIVE_BACKSPACE

            COMP    CHAR_COUNT,   0000
            RETURNZ
            LOAD    CURRENT_CASE, 0004
            LOAD    POINTER,      00A2
            LOAD    TEMP_REG,     0000
            OUTPUT  TEMP_REG,     0000
            SUB     CHAR_COUNT,   0001
            RETURN

RECEIVE_CRLF

            LOAD    CURRENT_CASE, 0005
            LOAD    POINTER,      00A5
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

```
            LOAD    TEMP_REG,       0000
            OUTPUT  TEMP_REG,       0000
            LOAD    CHAR_COUNT,     0000
            RETURN

RECIEVE_AT

            CALL    BinToASC
            LOAD    CURRENT_CASE, 0006
            LOAD    Pointer,        00A7
            LOAD    TEMP_REG,       0000
            OUTPUT  TEMP_REG,       0000
            LOAD    CHAR_COUNT,     0000
            RETURN

;=======================================================
;   BIN_TO_ASCII
;=======================================================
BinToASC
        LOAD    RE, CHAR_COUNT

        LOAD    RD, 000A
        CALL    FINDIT
        ADD     RB, 0030
        STORE   RB, 00A7

        ADD     RE, 0030
        STORE   RE, 00A8

        RETURN

;=======================================================
;   FInd It Routine
;=======================================================
FINDIT
        LOAD    RB, ZERO

REPEAT  SUB     RE, RD
        JUMPC   FOUNDIT
        ADD     RB, 0001
        JUMP    REPEAT
FOUNDIT
        ADD     RE, RD

        RETURN

;=======================================================
;   ISR
;=======================================================
        ADDRESS 0300
ISR
        INPUT   UART_STATUS,    PORT
        AND     UART_STATUS,    0003

        COMP    UART_STATUS,    0003
        JUMPZ   BOTH_RX_TX

        COMP    UART_STATUS,    TX_RDY
        CALLZ   TX

        COMP    UART_STATUS,    RX_RDY
        CALLZ   RX

        RETEN

BOTH_RX_TX

        CALL    TX
        CALL    RX

        RETEN

;========================================================
;   Main Loop
;   main loop is where processor spends most of its time
;========================================================
MAIN
        ; update LEDs
        JUMP    MAIN


;========================================================
;   ISR vectored through 0FFE
;========================================================
        ADDRESS 0FFE
ENDIT
        JUMP ISR

        END
```

Prepared by: Jose Sotelo	Date: May 14, 2018	Revision: 2.1

Appendix A: Key Terms

Start Bit

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To begin the transfer of data, the Tx pin pulls the transmission line from high too low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate selected.

Data Frame/Packet

The data frame contains the actual data being transferred. It can be 5 to 8 bits long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. In certain cases, the data sent can begin with the least significant bit first.

Parity

A parity bit is form of error checking that describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. If the parity bit is a 1 (odd parity), then there are an odd number of 1 bits in the data frame. If the parity bit is a 0 (even parity), then there are an even number of 1 bits in the data frame.

Stop Bit

To signal the end of the data transmission, the sending UART drives the data transmission line from a low voltage to a high voltage.

Baud Rate

The baud rate identifies the frequency the data is being transmitted at. (Bits per second)

Bit time

Bit time is the amount of time data bits are held on the wire.

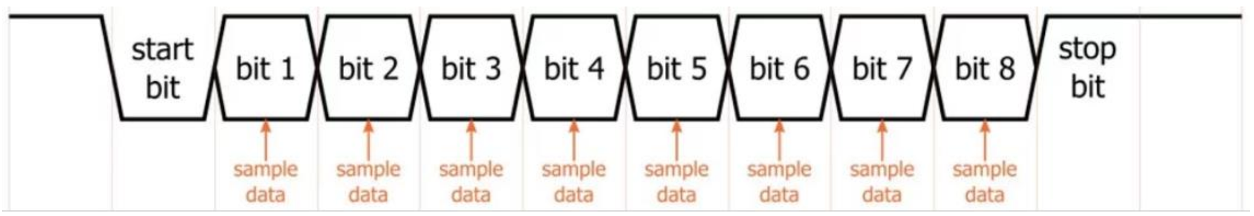


Figure : Illustration of Data Transmission