



CECS 460 SPRING 2019

UART with TSI Chip Specification

By

Raul Solorio



UART with TSI Chip Specification

Contents

1	INTRODUCTION	4
2	APPLICABLE DOCUMENTS.....	4
2.1	EXTERNAL DOCUMENTS.....	4
2.1.1	<i>PicoBlaze</i>	4
2.1.2	<i>ASCII Table</i>	6
2.1.3	<i>Artix-7 library guide</i>	7
3	REQUIREMENTS.....	7
4	TOP LEVEL DESIGN.....	8
4.1	DESCRIPTION.....	8
4.2	BLOCK DIAGRAM	8
4.3	DATA FLOW DESCRIPTION	9
4.4	I/O	9
4.4.1	<i>Signal Names</i>	9
4.4.2	<i>Pin Assignments</i>	10
4.4.3	<i>Electrical Characteristics</i>	10
4.4.4	<i>UART Configurations</i>	10
4.5	CLOCKS	10
5	EXTERNALLY DEVELOPED BLOCKS.....	11
5.1	DESCRIPTION.....	11
5.1.1	<i>tramelblaze_top</i>	11
5.1.2	<i>tramelblaze</i>	11
5.1.3	<i>stack_ram</i>	11
5.1.4	<i>scratch_ram</i>	11
5.1.5	<i>tb_rom</i>	11
5.1.6	<i>Register array</i>	11
5.2	BLOCK DIAGRAM	11
5.2.1	<i>tramelblaze_top</i>	11
5.3	I/O	12
6	INTERNALLY DEVELOPED BLOCKS	12
6.1	TSI.....	12
6.2	TOP_LEVEL.....	13
6.2.1	<i>Memory Map</i>	14
6.3	AISO.....	14
6.4	UART.....	15
6.5	BAUD_DECODE.....	16
6.6	RECEIVER	16
6.7	RX_STATE_MACHINE.....	17
6.8	TRANSMITTER	18
6.9	BIT_TIME_COUNTER	19
6.10	BIT_COUNTER.....	20
6.11	DECODE	20
6.12	SHIFT_REG	21



UART with TSI Chip Specification

6.13	PULSEMAKER	22
6.14	ADRS_DECODE	23
7	CHIP LEVEL VERIFICATION	24
8	CHIP LEVEL TEST	25



UART with TSI Chip Specification

1 Introduction

This Chip Specification document is intended to describe how the Universal Asynchronous Receiver Transmitter functions. My design was programmed using the tramelblaze.

Initially, the Transmitter was designed and interfaced with the tramelblaze allowing it to transmit data to a terminal. Building upon this, the Receiver was designed where both transmitter and receiver were instantiated to create the UART. Due to the time constraints, I was unable to fully implement the 32Kx16 SRAM. The Technology Specific Instantiations (TSI) instantiated all input and output buffers for the design.

UART is a chip that transfers serial data and receives parallel data. It allows a computer and other peripheral devices to communicate with each other and constantly checks for parity errors in case the data was not received correctly. The data can be executed when there is an interrupt at different speeds depending on the agreed baud rate between the devices.

2 Applicable Documents

2.1 External Documents

Since the PicoBlaze was no longer supported on the Nexys 4 board, Professor John Tramel created the TramelBlaze which was a 16-bit emulator of the 8-bit PicoBlaze. These documents are references on how the TramelBlaze works and how to implement them into our design.

2.1.1 PicoBlaze

The PicoBlaze embedded microcontroller is an efficient processor core for Xilinx FPGA's.

2.1.1.1 Architecture

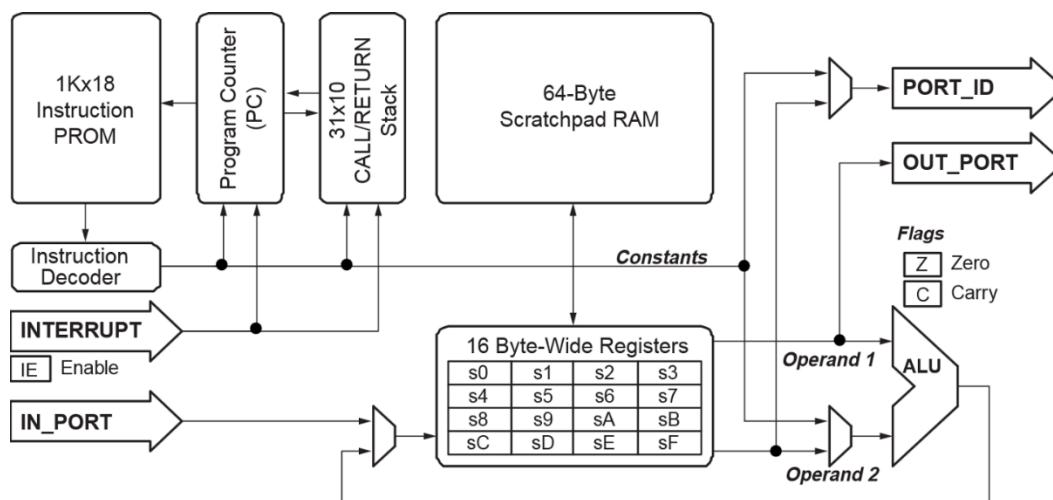


Figure 1: PicoBlaze Architecture



UART with TSI Chip Specification

The main difference between the TramelBlaze and PicoBlaze architecture is the size of the ROM and RAM. TramelBlaze has a 4Kx16 Instruction ROM, 128x16 CALL/RETURN Stack RAM, and 512x16 Scratchpad RAM. This schematic allows us to see how data is moving within the TramelBlaze to fully accomplish the UART.

2.1.1.2 Instruction Set

Instruction	Function		
0 NOP		29 RETURN	PC ← TOS+1
1 ADD rX, kk	rX ← rX + kk	30 RETURNC	if CARRY PC ← TOS+1
2 ADD rX, rY	rX ← rX + rY	31 RETURNNC	if !CARRY PC ← TOS+1
3 ADDCY rX, kk	rX ← rX + kk + CARRY	32 RETURNZ	if ZERO PC ← TOS+1
4 ADDCY rX, rY	rX ← rX + rY + CARRY	33 RETURNNZ	if !ZERO PC ← TOS+1
5 AND rX, kk	rX ← rX & kk	34 RETDIS	PC ← TOS+1; Interrupt Enable ← 0
6 AND rX, rY	rX ← rX & rY	35 RETEN	PC ← TOS+1; Interrupt Enable ← 1
7 CALL aaa	TOS ← PC; PC ← aaa	36 RL rX	rX ← {rX[14:0], rX[15]}; CARRY ← rX[15]
8 CALLC aaa	if CARRY TOS ← PC; PC ← aaa	37 RR rX	rX ← {rX[0], rX[15:1]}; CARRY ← rX[0]
9 CALLNC aaa	if !CARRY TOS ← PC; PC ← aaa	38 SLO rX	rX ← {rX[14:0], 0}; CARRY ← rX[15]
10 CALLZ aaa	if ZERO TOS ← PC; PC ← aaa	39 SLI rX	rX ← {rX[14:0], 1}; CARRY ← rX[15]
11 CALLNZ aaa	if !ZERO TOS ← PC; PC ← aaa	40 SLA rX	rX ← {rX[14:0], CARRY}; CARRY ← rX[15]
12 COMP rX, kk	If rX == kk then ZERO = 1; if rX < kk then CARRY = 1	41 SLX rX	rX ← {rX[14:0], rX[0]}; CARRY ← rX[15]
13 COMP rX, rY	If rX == rY then ZERO = 1; if rX < rY then CARRY = 1	42 SRQ rX	rX ← {0, rX[15:1]}; CARRY ← rX[0]
14 DISINT	Interrupt Enable = 0	43 SRI rX	rX ← {1, rX[15:1]}; CARRY ← rX[0]
15 EMINT	Interrupt Enable = 1	44 SRA rX	rX ← {CARRY, rX[15:1]}; CARRY ← rX[0]
16 INPUT rX, {rY}	rX ← PORT(rY)	45 SRX rX	rX ← {rX[15], rX[15:1]}; CARRY ← rX[0]
17 INPUT rX, pp	rX ← PORT(pp)	46 SUB rX, kk	rX ← rX - kk
18 JUMP aaa	PC ← aaa	47 SUB rX, rY	rX ← rX - rY
19 JUMPC aaa	if CARRY PC ← aaa	48 SUBC rX, kk	rX ← rX - kk - CARRY
20 JUMENC aaa	if !CARRY PC ← aaa	49 SUBC rX, rY	rX ← rX - rY - CARRY
21 JUMPZ aaa	if ZERO PC ← aaa	50 TEST rX, kk	If (rX & kk) == 0; ZERO ← 1; CARRY ← ODD PARITY (rX & kk)
22 JUMPNZ aaa	if !ZERO PC ← aaa	51 TEST rX, rY	If (rX & rY) == 0; ZERO ← 1; CARRY ← ODD PARITY (rX & rY)
23 LOAD rX, kk	rX ← kk	52 NOR rX, kk	rX ← rX ^ kk
24 LOAD rX, rY	rX ← rY	53 XOR rX, rY	rX ← rX ^ rY
25 OR rX, kk	rX ← rX kk	54 FETCH rX, kk	rX ← RAM[kk]
26 OR rX, rY	rX ← rX rY	55 FETCH rX, {rY}	rX ← RAM[{rY}]
27 OUTPUT rX, {rY}	PORT(rY) ← rX	56 STORE rX, kk	RAM[kk] ← rX
28 OUTPUT rX, pp	PORT(pp) ← rX	57 STORE rX, {rY}	RAM[{rY}] ← rX

Figure 2: Instruction Set

These instructions are used within the assembly code to program the PicoBlaze. They are also applicable to the TramelBlaze.



UART with TSI Chip Specification

2.1.1.3 READ/WRITE TIMING

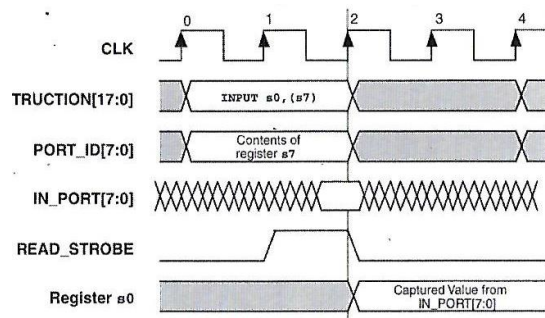


Figure 3: READ TIMING

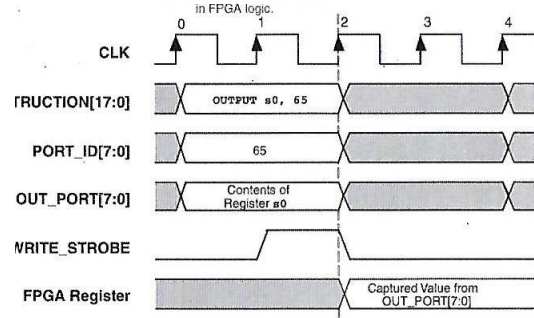


Figure 4: WRITE TIMING

The microcontroller captures the read or write value after two clock cycles. The INPUT data is being read by the PicoBlaze/TramelBlaze and OUTPUT data is being sent by the PicoBlaze/TramelBlaze into our UART.

2.1.2 ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	##32;	Space	64	40	100	##64;	@	96	60	140	##96;	`
1	1	001	SOH (start of heading)	33	21	041	##33;	!	65	41	101	##65;	A	97	61	141	##97;	a
2	2	002	STX (start of text)	34	22	042	##34;	"	66	42	102	##66;	B	98	62	142	##98;	b
3	3	003	ETX (end of text)	35	23	043	##35;	#	67	43	103	##67;	C	99	63	143	##99;	c
4	4	004	END (end of transmission)	36	24	044	##36;	\$	68	44	104	##68;	D	100	64	144	##100;	d
5	5	005	ENQ (enquiry)	37	25	045	##37;	%	69	45	105	##69;	E	101	65	145	##101;	e
6	6	006	ACK (acknowledge)	38	26	046	##38;	&	70	46	106	##70;	F	102	66	146	##102;	f
7	7	007	BEL (bell)	39	27	047	##39;	'	71	47	107	##71;	G	103	67	147	##103;	g
8	8	010	BS (backspace)	40	28	050	##40;	(72	48	110	##72;	H	104	68	150	##104;	h
9	9	011	TAB (horizontal tab)	41	29	051	##41;)	73	49	111	##73;	I	105	69	151	##105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	##42;	*	74	4A	112	##74;	J	106	6A	152	##106;	j
11	B	013	VT (vertical tab)	43	2B	053	##43;	+	75	4B	113	##75;	K	107	6B	153	##107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	##44;	,	76	4C	114	##76;	L	108	6C	154	##108;	l
13	D	015	CR (carriage return)	45	2D	055	##45;	-	77	4D	115	##77;	M	109	6D	155	##109;	m
14	E	016	SO (shift out)	46	2E	056	##46;	.	78	4E	116	##78;	N	110	6E	156	##110;	n
15	F	017	SI (shift in)	47	2F	057	##47;	/	79	4F	117	##79;	O	111	6F	157	##111;	o
16	10	020	DLE (data link escape)	48	30	060	##48;	0	80	50	120	##80;	P	112	70	160	##112;	p
17	11	021	DC1 (device control 1)	49	31	061	##49;	1	81	51	121	##81;	Q	113	71	161	##113;	q
18	12	022	DC2 (device control 2)	50	32	062	##50;	2	82	52	122	##82;	R	114	72	162	##114;	r
19	13	023	DC3 (device control 3)	51	33	063	##51;	3	83	53	123	##83;	S	115	73	163	##115;	s
20	14	024	DC4 (device control 4)	52	34	064	##52;	4	84	54	124	##84;	T	116	74	164	##116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	##53;	5	85	55	125	##85;	U	117	75	165	##117;	u
22	16	026	SYN (synchronous idle)	54	36	066	##54;	6	86	56	126	##86;	V	118	76	166	##118;	v
23	17	027	ETB (end of trans. block)	55	37	067	##55;	7	87	57	127	##87;	W	119	77	167	##119;	w
24	18	030	CAN (cancel)	56	38	070	##56;	8	88	58	130	##88;	X	120	78	170	##120;	x
25	19	031	EM (end of medium)	57	39	071	##57;	9	89	59	131	##89;	Y	121	79	171	##121;	y
26	1A	032	SUB (substitute)	58	3A	072	##58;	:	90	5A	132	##90;	Z	122	7A	172	##122;	z
27	1B	033	ESC (escape)	59	3B	073	##59;	;	91	5B	133	##91;	[123	7B	173	##123;	{
28	1C	034	FS (file separator)	60	3C	074	##60;	<	92	5C	134	##92;	\	124	7C	174	##124;	
29	1D	035	GS (group separator)	61	3D	075	##61;	=	93	5D	135	##93;]	125	7D	175	##125;	}
30	1E	036	RS (record separator)	62	3E	076	##62;	>	94	5E	136	##94;	^	126	7E	176	##126;	~
31	1F	037	US (unit separator)	63	3F	077	##63;	?	95	5F	137	##95;	_	127	7F	177	##127;	DEL

Source: www.LookupTables.com

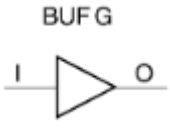
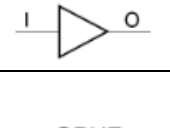

Figure 5: ASCII TABLE



UART with TSI Chip Specification

2.1.3 Artix-7 library guide

The Artix-7 Library guide provides information on how to instantiate the buffers and what each buffer is used for. The ones used in my design for the TSI includes BUFG, IBUF, and OBUF.

Diagram	Description
	High-fanout buffer that connects signals to the global routing resources for low skew distribution of signal. Typically used on clocks.
	Input port is connected directly to the associated top-level input and the output is connected to the logic sourced by that port.
	Provides drive current for signals leaving a chip. Its output is connected to an OPAD/IOPAD. The interface standard used by this is LVCMOS18. We can also select the drive and slew rates using DRIVE, SLOW, or FAST constraints.

3 Requirements

An LED on the board should be rotating or shifting left to indicate we are within our main loop. As soon as the transmitter is ready to transmit data, it displays a banner followed by a prompt. Once the prompt is displayed it should wait for the receiver to be ready to receive data. As we enter a key on our computer, the data goes to our receiver and it should echo (transmit) back the data entered. When the character '*' is pressed it displays my hometown then the prompt, '@' displays the number of characters received since reset, then the prompt (the characters '@', '*', 'BS', 'CR' do not increment the counter). When backspace is entered it should not delete the prompt but should delete the character entered before and 'CR' displays a new prompt. Implementing the TSI should not change the software requirements and if the cells were instantiated correctly, then the data should be able to display.

Although the SRAM was not implemented into this design, the software requirements are the following. Before displaying the banner and prompt, there should be a memory test by writing AAAA's, 5555's, Address, and Address bar into SRAM. Once finished, it should display a memory pass or fail indicating the SRAM was instantiated correctly. Every time we receive a byte from the UART we store it into the SRAM and when it receives the character '%' it dumps the data within the SRAM into the terminal but the data inside the memory block never gets cleared.



UART with TSI Chip Specification

4 Top Level Design

4.1 Description

The overall top-level design consists of the Core and TSI where they are both instantiated and connected. The Core consists of the AISO, TramelBlaze, UART, and ADRS_DEC which makes up the Full UART. If we were to implement the SRAM, it would be instantiated within the Core. The TSI contains all references to the target technology library so all input and output signal needs to go through the TSI before communicating with each other.

4.2 Block Diagram

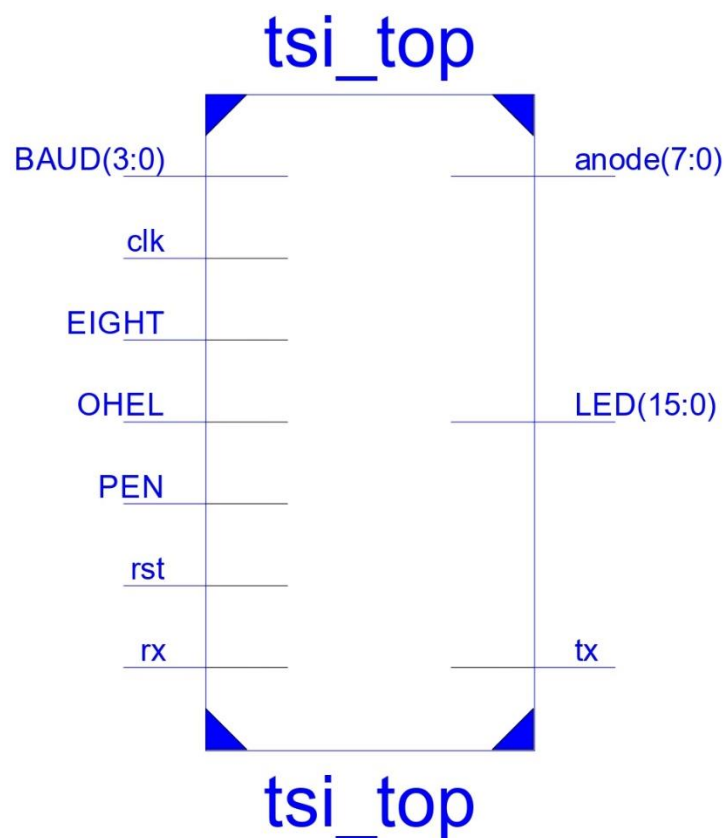


Figure 6: Overall Top Level Module



UART with TSI Chip Specification

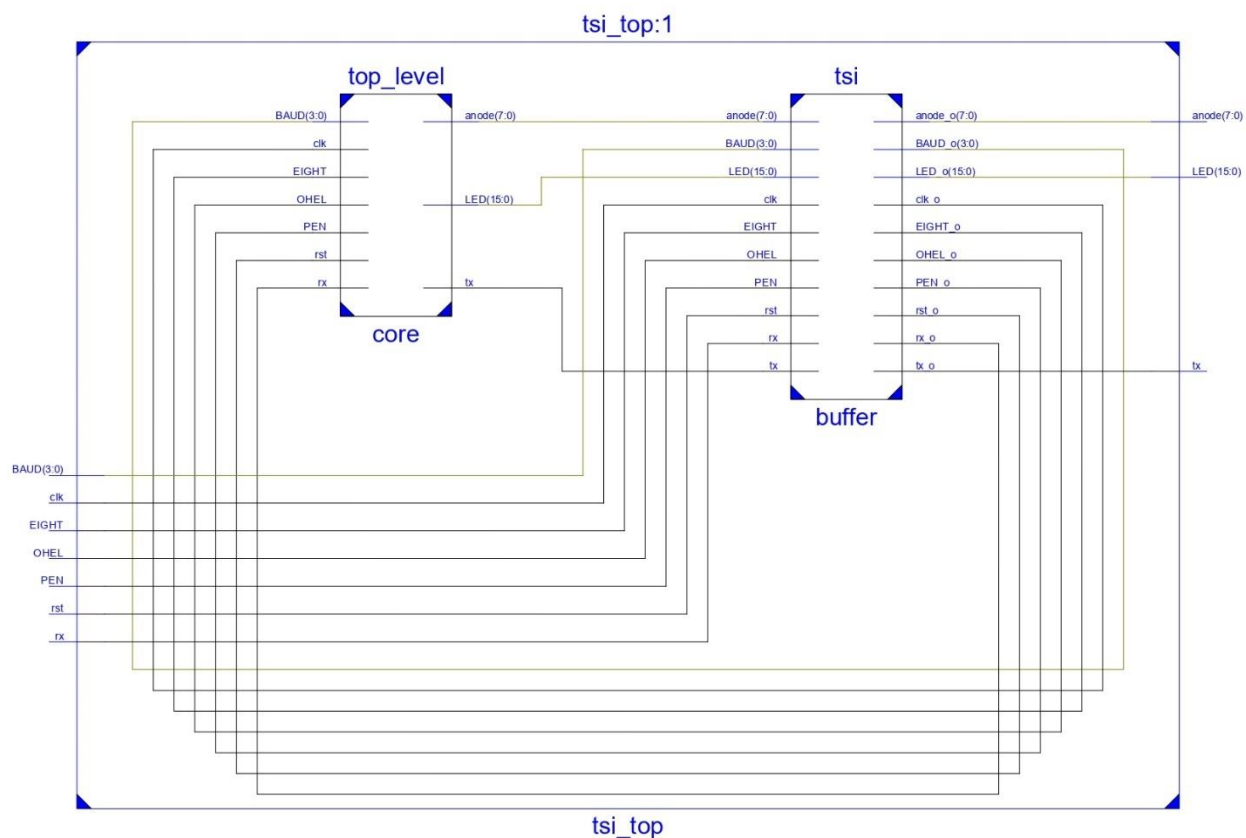


Figure 7: Detailed Block Diagram

4.3 Data Flow Description

The input signals pass through the buffers and exit into the core while the output signal from the core passes through the buffers and exits into the board.

4.4 I/O

4.4.1 Signal Names

clk	LED[0]	LED[10]	anode[4]
OHEL	LED[1]	LED[11]	anode[5]
PEN	LED[2]	LED[12]	anode[6]
EIGHT	LED[3]	LED[13]	anode[7]
BAUD[0]	LED[4]	LED[14]	tx
BAUD[1]	LED[5]	LED[15]	
BAUD[2]	LED[6]	anode[0]	
BAUD[3]	LED[7]	anode[1]	
rst	LED[8]	anode[2]	
rx	LED[9]	anode[3]	



UART with TSI Chip Specification

4.4.2 Pin Assignments

Input Signal	Pin Location	Output Signal	Pin Location	Output Signal	Pin Location	Output Signal	Pin Location
clk	E3	LED[0]	H17	LED[10]	U14	anode[4]	P14
OHEL	L16	LED[1]	K15	LED[11]	T16	anode[5]	T14
PEN	M13	LED[2]	J13	LED[12]	V15	anode[6]	K2
EIGHT	R15	LED[3]	N14	LED[13]	V14	anode[7]	U13
BAUD[0]	R17	LED[4]	R18	LED[14]	V12	tx	D4
BAUD[1]	T18	LED[5]	V17	LED[15]	V11		
BAUD[2]	U18	LED[6]	U17	anode[0]	J17		
BAUD[3]	R13	LED[7]	U16	anode[1]	J18		
rst	M18	LED[8]	V16	anode[2]	T9		
rx	C4	LED[9]	T15	anode[3]	J14		

4.4.3 Electrical Characteristics

Switches, buttons, and 7-seg display have an input of 3.3V

4.4.4 UART Configurations

CASE	RATE	BIT TIME	N4 COUNT
0000	300	3.3333ms	333,333
0001	1200	833.33us	83,333
0010	2400	416.66us	41,667
0011	4800	208.33us	20,833
0100	9600	104.16us	10,417
0101	19200	52.083us	5,208
0110	38400	26.041us	2,604
0111	57600	17.361us	1,736
1000	115200	8.6806us	868
1001	230400	4.3403us	434
1010	460800	2.1701us	217
1011	921600	1.0851us	109

Our 4-bit BAUD signal indicates which baud rate we would like to use for our design. The table above translates these bits into their appropriate rate, time, and equivalent Verilog count value.

Signal	Description
OHEL	Checks if we want an even or low parity
PEN	Enables parity
EIGHT	Indicates if we want to send 8 bits or 7 bits

4.5 Clocks

Nexys 4 uses an 100MHz crystal oscillator Located in pin E3.

4.6 Reset

The reset signal comes from one of the push buttons on the Nexys 4. It will be a high active asynchronous signal which will get synchronized once passed through the AISO.



UART with TSI Chip Specification

4.7 Software

Located in the appendix

4.7.1 Detailed software planning document

Located in the appendix

5 Externally Developed Blocks

5.1 Description

5.1.1 `tramelblaze_top`

instantiates the `tramelblaze` and creates rom.

5.1.2 `tramelblaze`

A 16-bit emulator of the 8-bit `picoblaze`.

5.1.3 `stack_ram`

128x18 ram.

5.1.4 `scratch_ram`

512x16 ram.

5.1.5 `tb_rom`

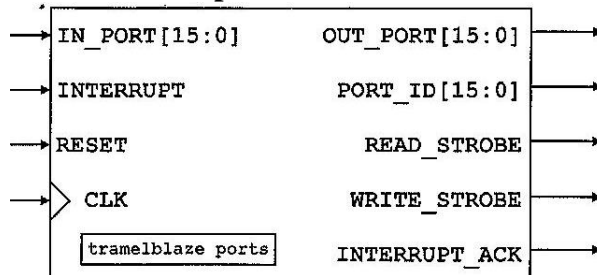
4096x16 rom.

5.1.6 Register array

16x16 memory.

5.2 Block Diagram

5.2.1 `tramelblaze_top`





UART with TSI Chip Specification

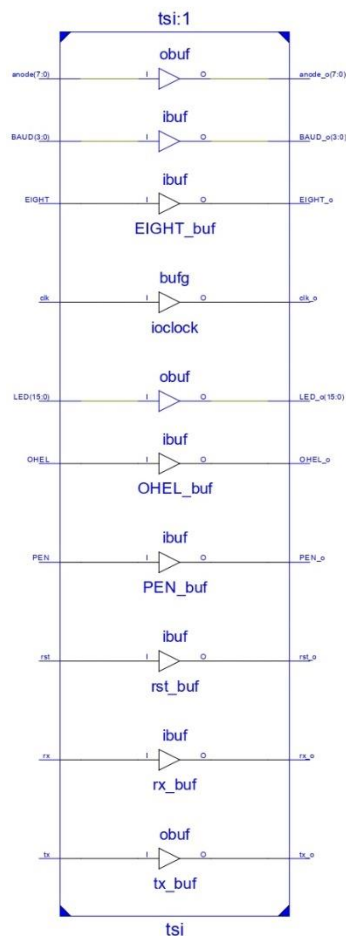
5.3 I/O

INPUT SIGNAL	OUTPUT SIGNAL
IN_PORT[15:0]	OUT_PORT[15:0]
INTERRUPT	PORT_ID[15:0]
RESET	READ_STROBE
CLK	WRITE_STROBE
	INTERRUPT_ACK

6 Internally Developed Blocks

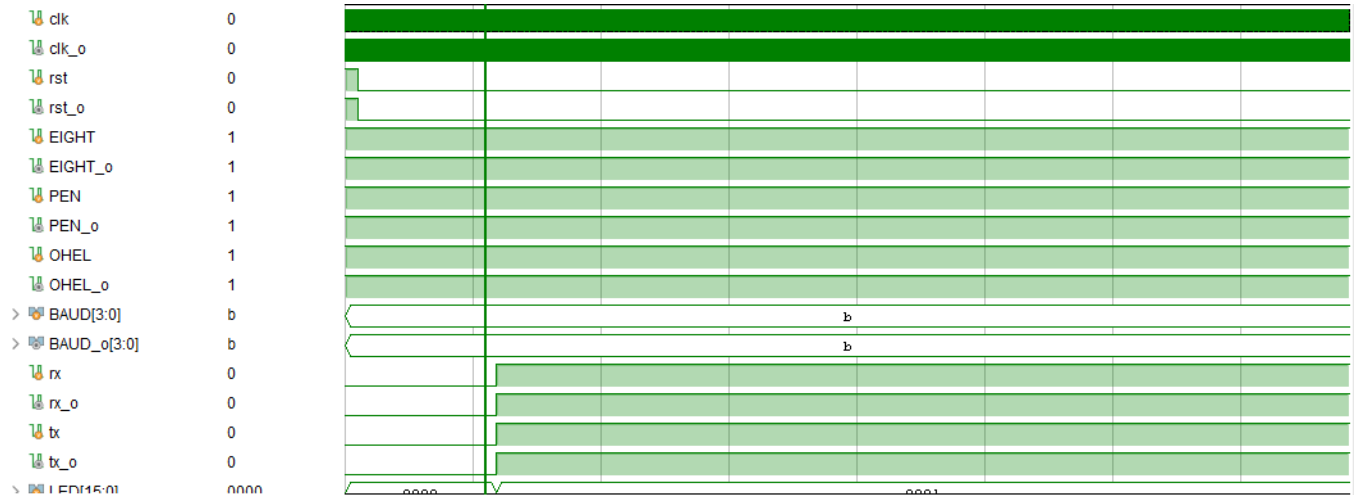
6.1 tsi

This module contains all references to the target technology library. Every Input and Output signal passes through here. I used specifically used BUFG, IBUF, and OBUF.





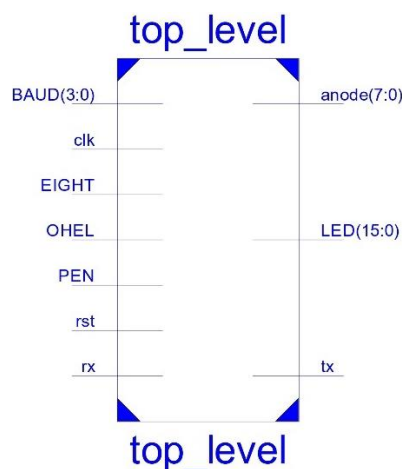
UART with TSI Chip Specification



TSI was verified by sending some signals into the module and made sure the output signals of the buffer followed the input signals.

6.2 top_level

Interconnect's all modules together by instantiating the appropriate modules and connects them with wires. I designed the loadable register so the out_port data of the tramelblaze gets written to the LED's depending on the address decoder. This feedback allowed me to verify my software was running in the main loop of my Assembly Code. Each anode was sent a bit value of '1' to turn them off. If this is not done, the anodes will be dimly on even though they are not used in the UCF file. Four inputs will be given as the baud rate and another three inputs will be used to determine the parity protocol. To see the data coming out of tx, connect it to a computer and within the program "Realterm", we can display the data being sent. The rx input signal is the port we are receiving data. The UART works with the tramelblaze to transmit and receive the data at the appropriate time.





UART with TSI Chip Specification

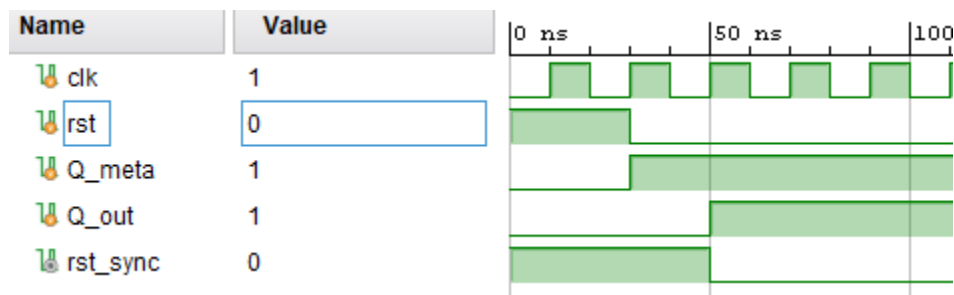
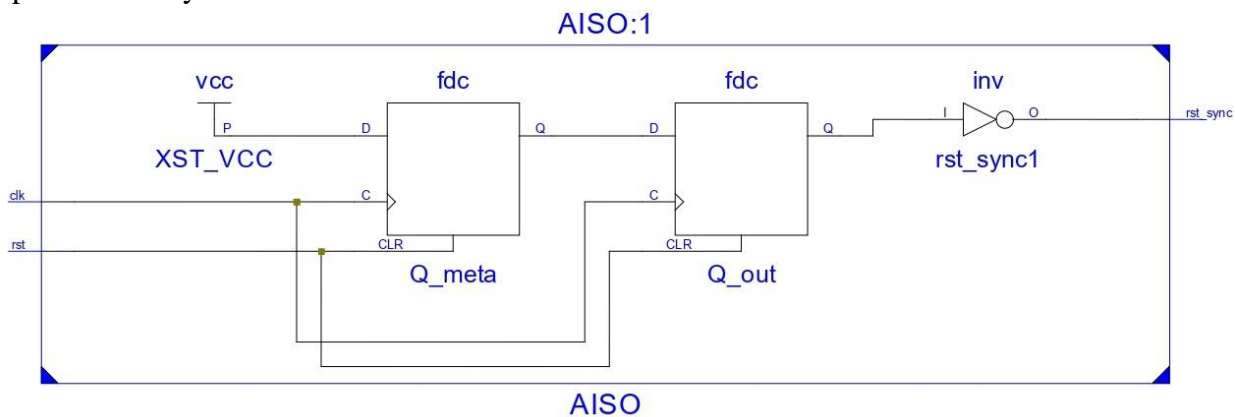
6.2.1 Memory Map

Address	Contents
0x0000	UART Data Register (R/W)
0x0001	UART Status Register (RO)
0x0002	LED[15:0] (WO)
0x0003	SevenSegment[31:16] (WO)
0x0004	SevenSegment[15:0] (WO)
0x0005	Switches (RO)
0x0006	UART Configuration (R/W)

This memory map was used to determine where to write to or read from. The address decoder within sends the correct signal to the UART and top level to decide what to do with the content.

6.3 AISO

The AISO module generates a synchronous reset for resetting all logic in the design. The initial reset signal gets passed through two flip flops where one produces a metastable signal and the second one produces the stable signal which will get outputted. That signal gets inverted to produce the synchronous reset.



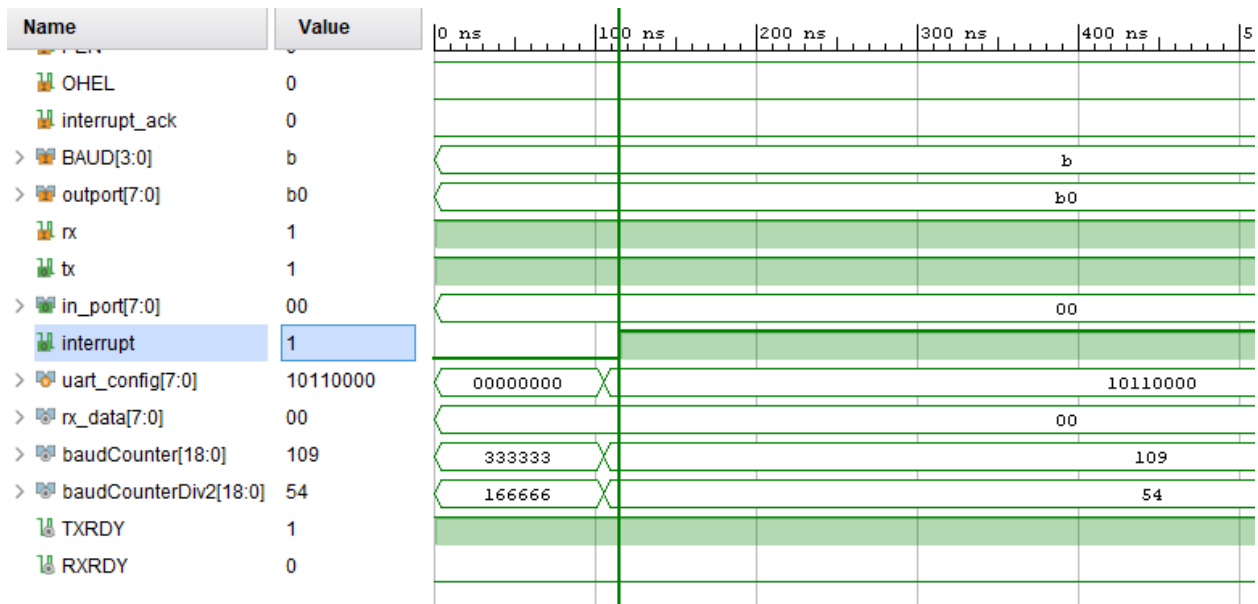
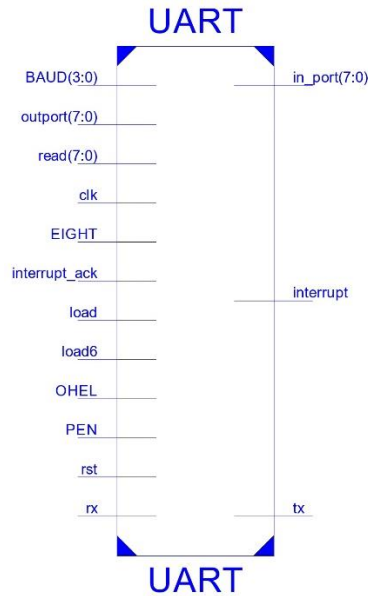
When reset is triggered, both outputs of the flops go inactive. The output of the synchronous reset signal is high since we inverted it. After reset is done, the metastable signal goes high, but the other signal stays low due to the timing delay in a flop thus the synchronous signal is still active. In the next clock cycle, the second flop becomes active, inverting the synchronous signal.



UART with TSI Chip Specification

6.4 UART

Main purpose is to transmit serial data and receive parallel data. This module takes in four inputs representing the selectable baud rate and gets converted into a counter value inside the baud decoder. The other inputs are used in the Transmitter which comes out of reset with the signal TXRDY active, meaning it is ready to transmit data. When data is being written to the UART, TXRDY is inactive. Once it is complete, the signal goes back to being active. The tx signal is the bit being sent out of the Transmitter. The receiver gets the counter value along with half of the counter value. RXRDY comes out of reset inactive and gets set once it has received all the data bits.



At reset TXRDY is active. I confirmed that my uart_config got the switches data. RXRDY comes out of reset as inactive. Since TXRDY is active, there is an interrupt signal.



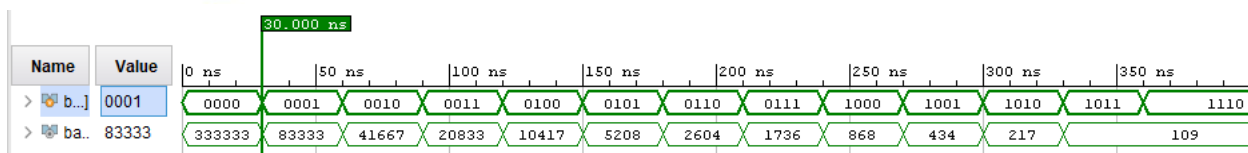
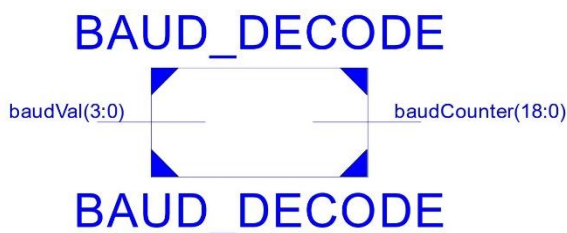
UART with TSI Chip Specification



As time passes by TXRDY should still be active since there is no data being transmitted however RXRDY is still inactive since it is receiving data. Once it finished receiving data RXRDY goes active meaning is ready to receive data.

6.5 BAUD_DECODE

It takes in a four-bit input representing a baud rate from range 300 to 921600. It goes through the decoder mapping its rate to the appropriate 19-bit count value. The count value is calculated based on its bit time and board being used. There are only 12 baud rate values in this UART, so any extra input is defaulted to 921600.



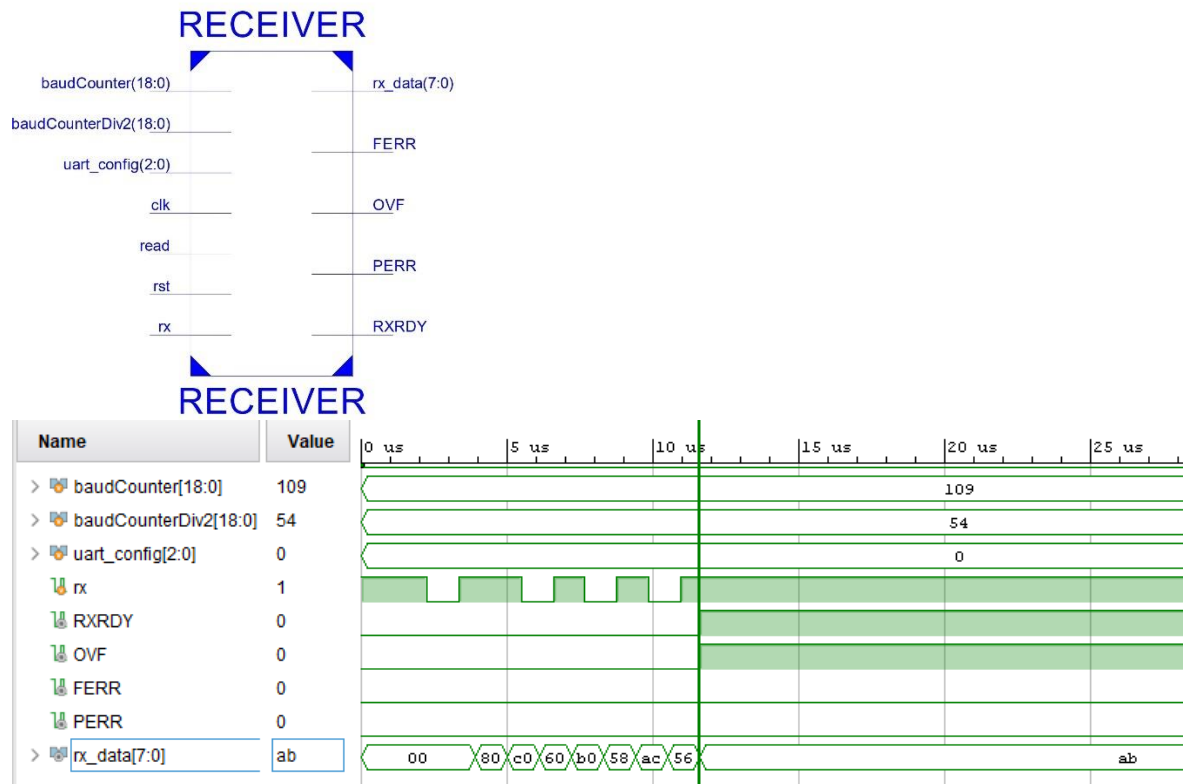
Each case of the 4-bit input was tried to get the appropriate baud rate. The last case is 1011 that gives a baud rate of 109. If, for example, 1110 is an input, the default gets set to 109.

6.6 RECEIVER

Receives data serially which gets converted to parallel data. It keeps track of our baud counter and makes sure we have no parity error. It also knows whether it gets 8 bits or 7 bits along with an Even, Odd or no parity bit. RXRDY comes out of reset inactive and gets set once it has received all the data bits.



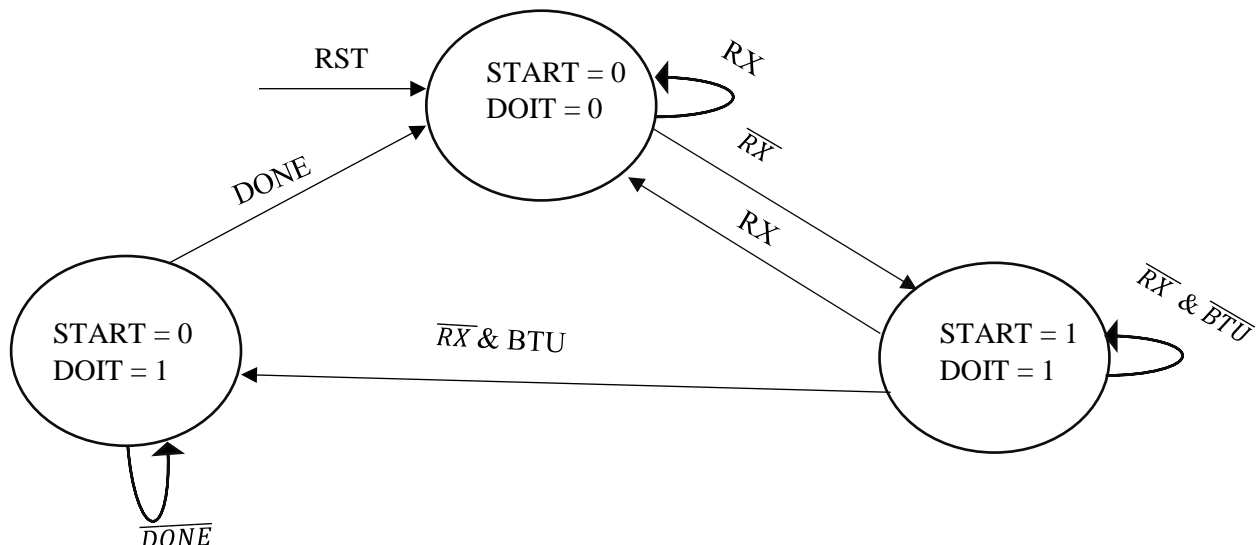
UART with TSI Chip Specification



As the receive engine is getting the data, the RXRDY is inactive. Once we received the last bit RXRDY goes active indicating it is ready to receive data again. Data being sent is 0xAB with 7 bits, no parity at a baud rate of 921600 but gets truncated to 0x2B due to it being 7 bits. The data within rx_data is being shifted and finally received data 0xab.

6.7 rx_state_machine

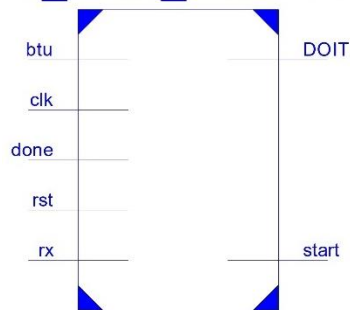
The state machine outputs the START and DOIT bit. At reset if we don't have a start bit '1', then we need to count half of the overall baud time. Once we have reached the time and the incoming bit is a '0' then start goes to '0' so the actual baud time gets counted. Once we have finished receiving data, then both START and DOIT go inactive. These signals tell us when to count our time and bits along with reading the data when it is in the middle of time.



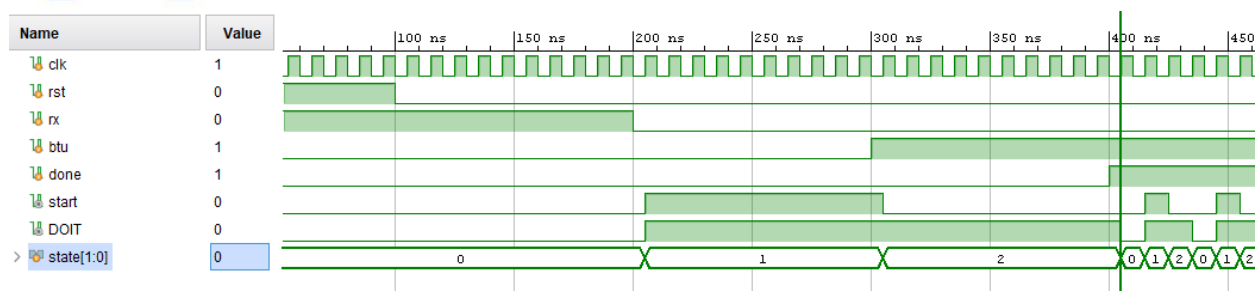


UART with TSI Chip Specification

rx_state_machine



rx_state_machine

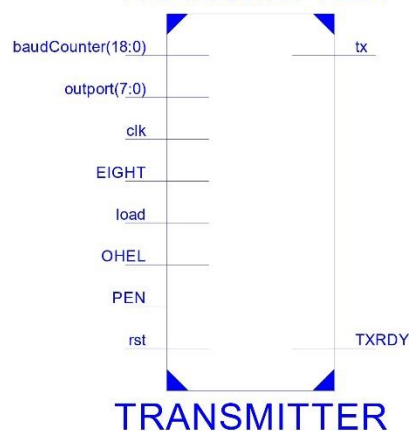


At reset we are at state 0 and when rx goes high it stays at state 0. When rx is low, meaning our start bit is being received, it goes to state 1. When rx is left low and we have not reached the baud rate, then stay at state 1. Once it reached the baud rate it goes to state 2. When done is inactive it stays at state 2 and when it goes active it goes to state 0.

6.8 TRANSMITTER

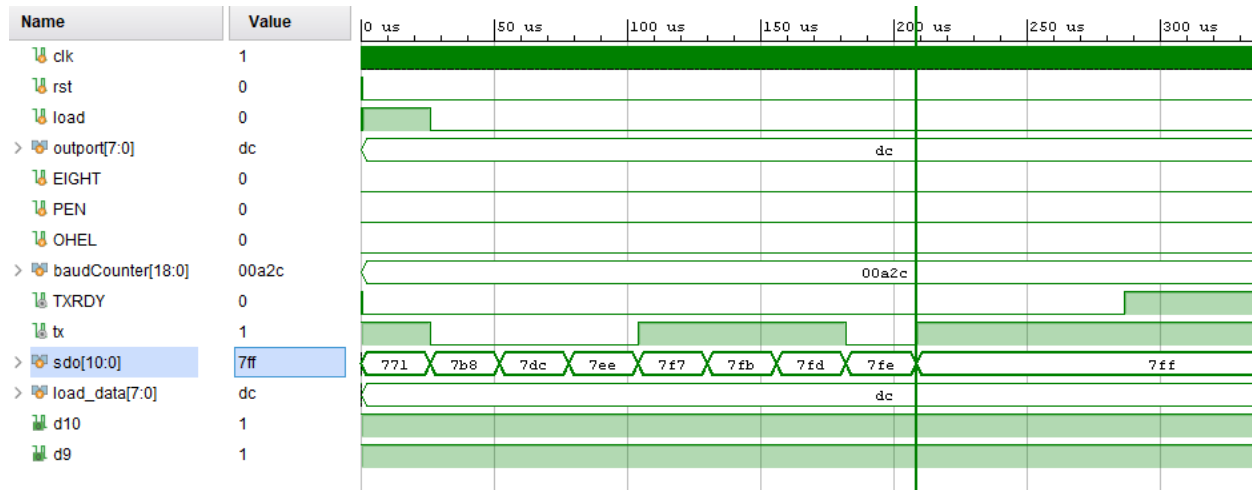
The purpose of this device is to output the correct information serially and indicate whether it is ready to send the data or wait until data is finished writing. As it is outputting each bit, the machine keeps track of how long each bit will be held and how many bits were sent out. The data coming in is decoded to generate the missing bits it is going to transmit. This will follow the parity protocol.

TRANSMITTER





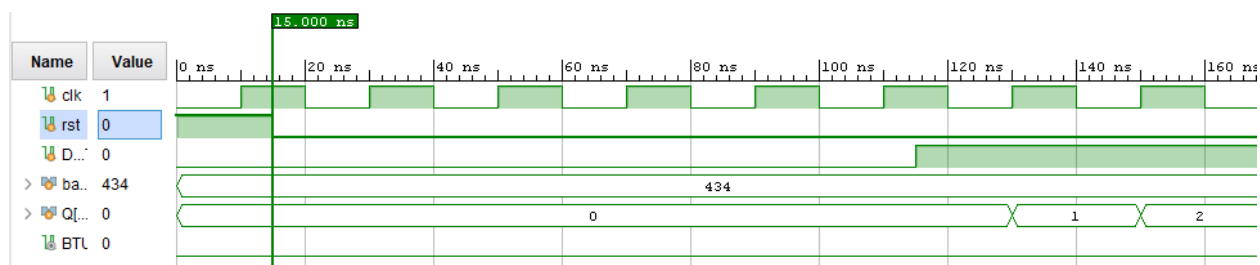
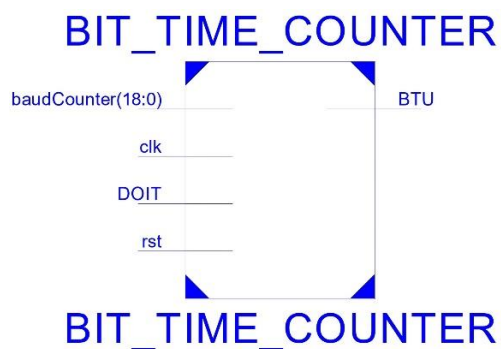
UART with TSI Chip Specification



I verified the transmit engine was active at reset and it read the correct baud rate going into the UART. As time passed by and data got written into the transmitter, 'TXRDY' was inactive and the data was shifting data 0xDC right were tx got the lsb of the shift register. Since I verified it with our protocol of 7 bits, no parity then the signal d10 and d9 become active.

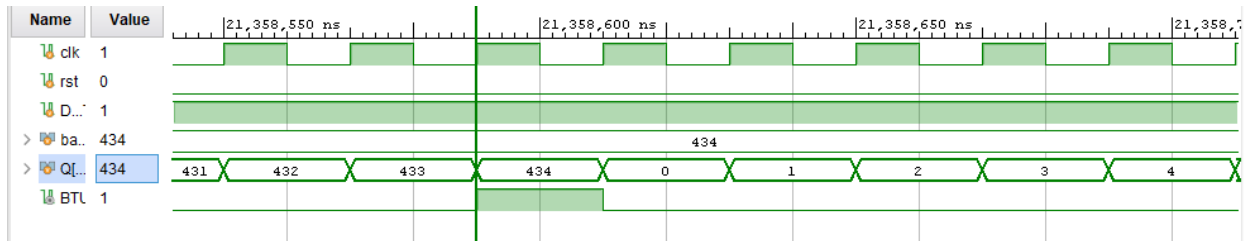
6.9 BIT_TIME_COUNTER

When data is being written and our counter has not reached the baud rate, the counter will be incremented by one. Once it is done writing or counter has reached the baud rate, the counter gets set back to zero. This module sends a signal indicating it has reached the baud rate which is basically how long each bit is held on the wire.





UART with TSI Chip Specification

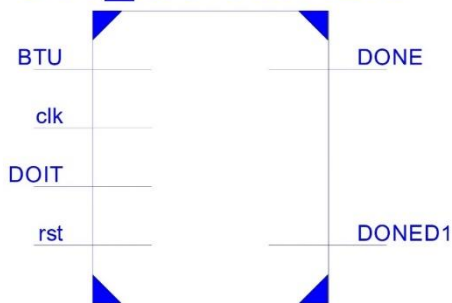


At reset, counter gets a value of 0 and BTU outputs a '0' since our counter has not reached our baud counter of 434. Once our counter reached that baud rate, BTU outputs a '1'.

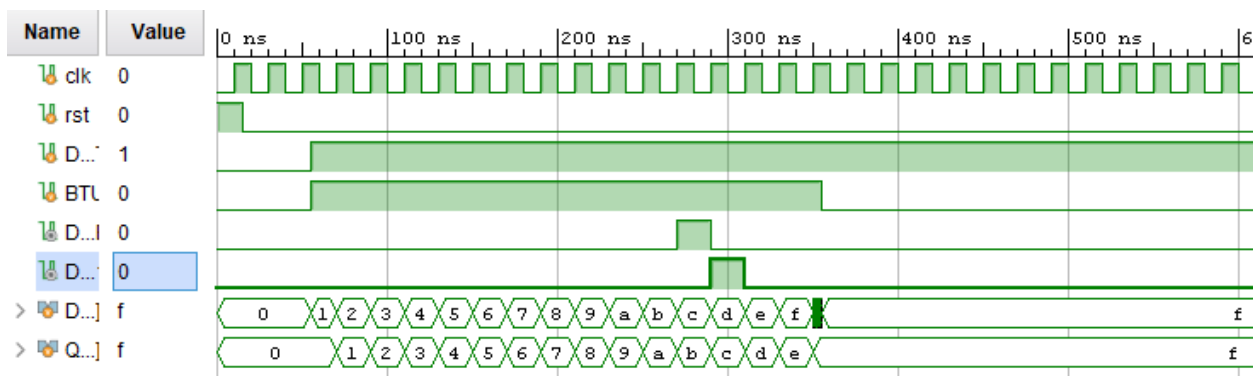
6.10 BIT_COUNTER

This module allows the engine to keep track of the bits sent, which for this project it is eleven. When data is being written and we have reached our baud rate, the bit counter will increment by one. The DONED1 signal is used in the overall transmitter to indicate it is ready to send data again.

BIT_COUNTER



BIT_COUNTER



At reset, our bit counter gets set to '0'. When both DOIT and BTU signal are active, the counter starts incrementing. Once our counter reaches 11 or hex value 'b', DONE and DONED1 go active.

6.11 Decode

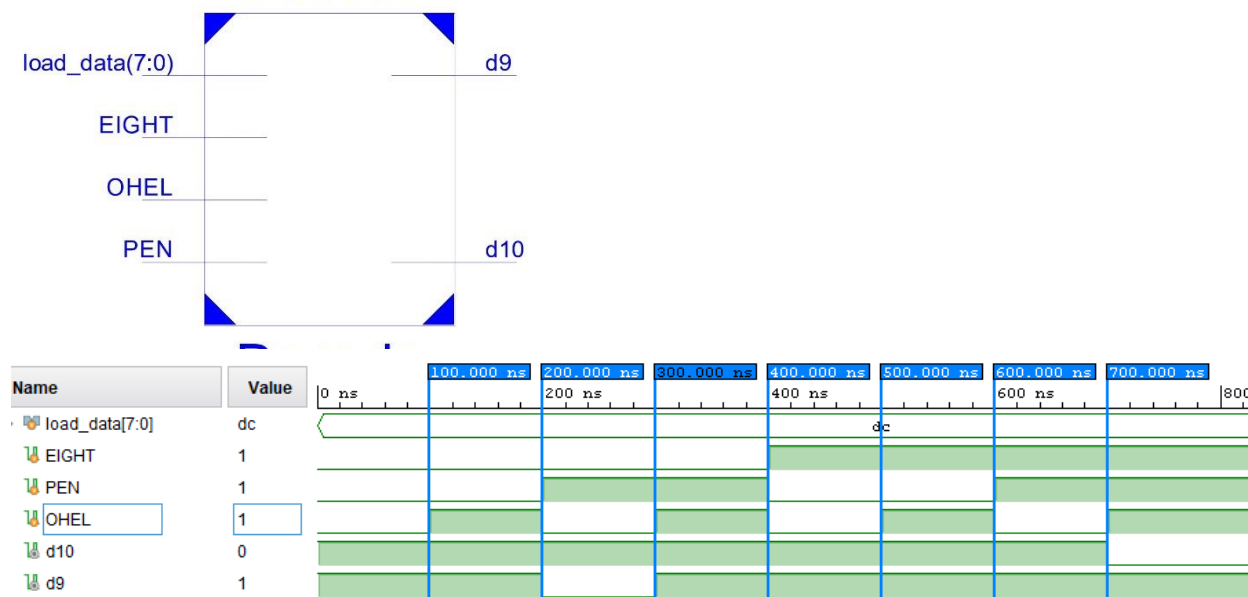
The decode module generates the missing bits to be transmitted depending on the given inputs. It first obtains the 8-bit data and xor's the 8 bits to determine the parity bit value. Seven bits are then xor'ed to determine the parity bit value when given seven bits as a configuration.



UART with TSI Chip Specification

Based on the given inputs, it will output a two-bit value containing a 1, the even or odd parity, or a bit from the loaded data. This varies by case.

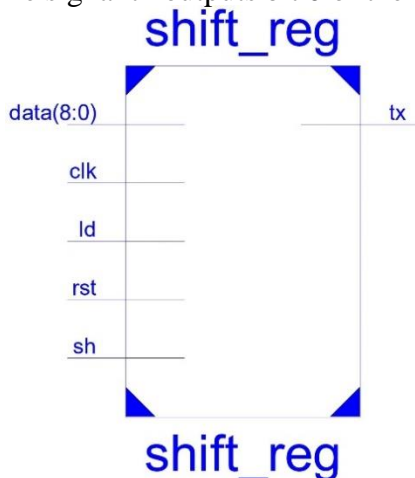
Decode



Each one of the markers marks the end of the case. When all inputs are 0's both or we have `OHEL` as the only one active then both outputs go active. When we have (7E1) or 7 bits even parity enabled then the MSB stayed active while `d9` became inactive due to the even bit. When we change it to odd parity then both outputs go active. Once it configures to 8 bits no parity outputs remain active and gives the same result for 8 bits with even parity. Once all input signal is high meaning (801), then the MSB goes inactive due to the even parity and `d9` stays active. The data being sent throughout the simulation was 0xDC

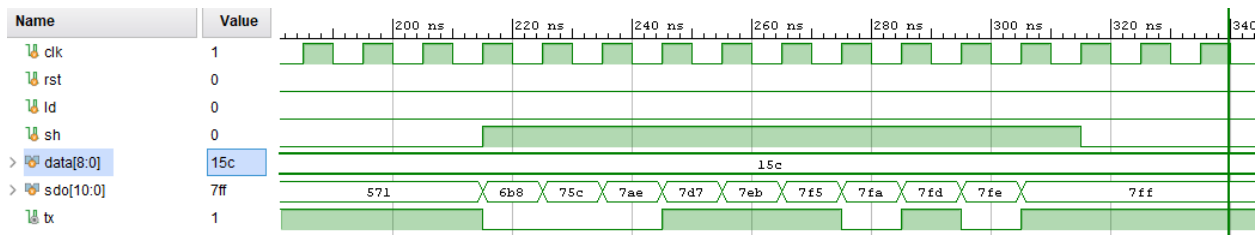
6.12 shift_reg

The purpose of this module is to correctly send the information out. At reset it gets assigned to an inactive state of '1' and that data gets transmitted out serially. Once data gets loaded in, it does a right shift with the LSB being transmitted out and the MSB fill with a '1'. The signal tx outputs bit 0 of the data.





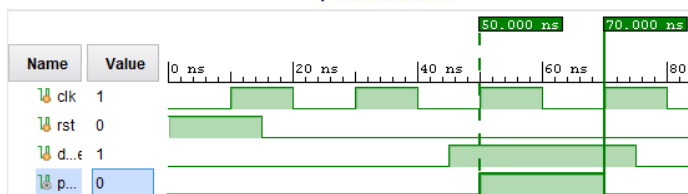
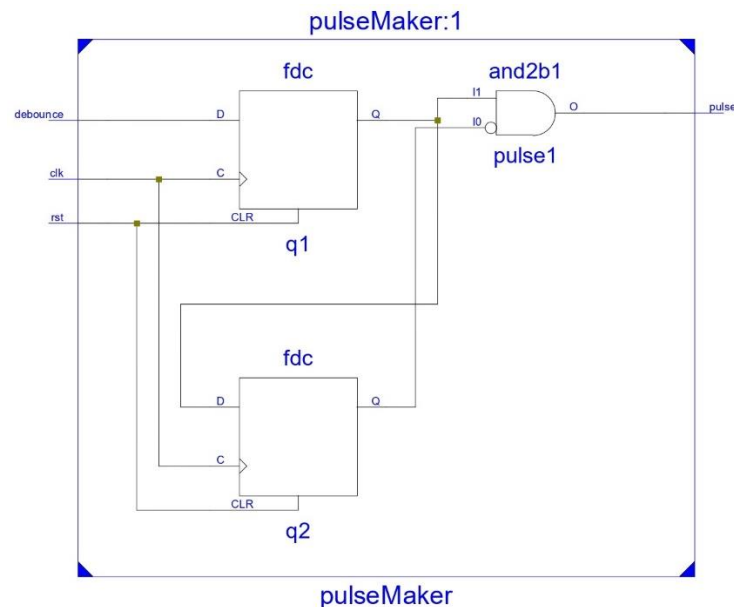
UART with TSI Chip Specification



I verified my shift register by trying out data 0x15C. At reset the data on the register sdo is 11 bit 1's. Since bit 0 is a '1', tx signal is active. Once data 0x15C got loaded into the register, it got concatenated with a '0' and a '1' creating the value 0x571. Once again, tx signal is kept active due to the LSB being a '1'. When 'sh' is active, data gets shifted to the right outputting the LSB on the tx.

6.13 pulseMaker

This module detects a positive edge from the debounce module and generates a pulse that last one clock period. This signal gets passed through two flops, where the first one outputs an unstable signal and the second one outputs the stable signal. The output of the first flop passes through an 'and' gate while the output of the second flop gets negated and passes through the 'and' gate as well.



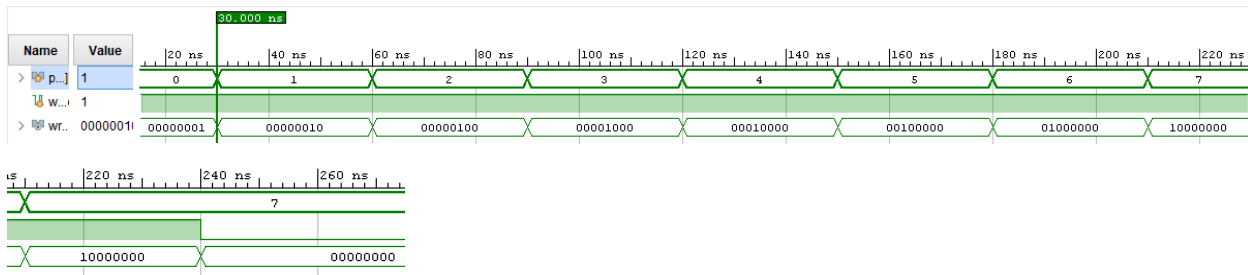
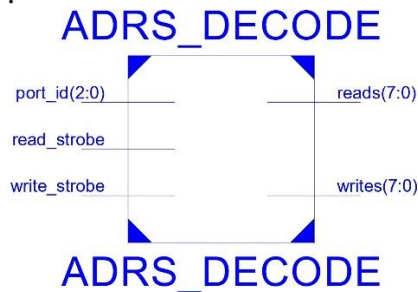
At reset, the signal gets set to '0'. Once we have a debounce signal, pulse will go active for one clock period, in this case 20ns.



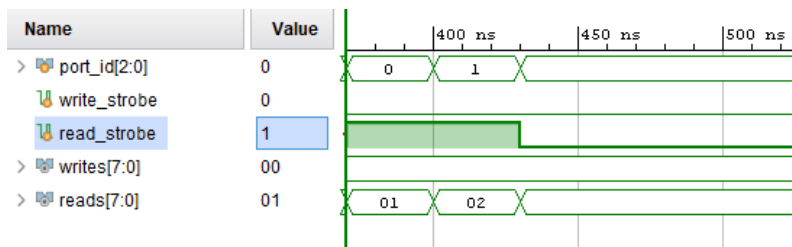
UART with TSI Chip Specification

6.14 ADRS_DECODE

Our port_id signal is our address which we would like to read data from or write data to base on our write and read strobe. This module then decodes the address and sets the appropriate bit based on the address we read or write.



Every case of the port_id was tried with the write_strobe being active. In each case, we see writes changing bits. Once write_strobe went inactive, writes got 8-bit zeros.

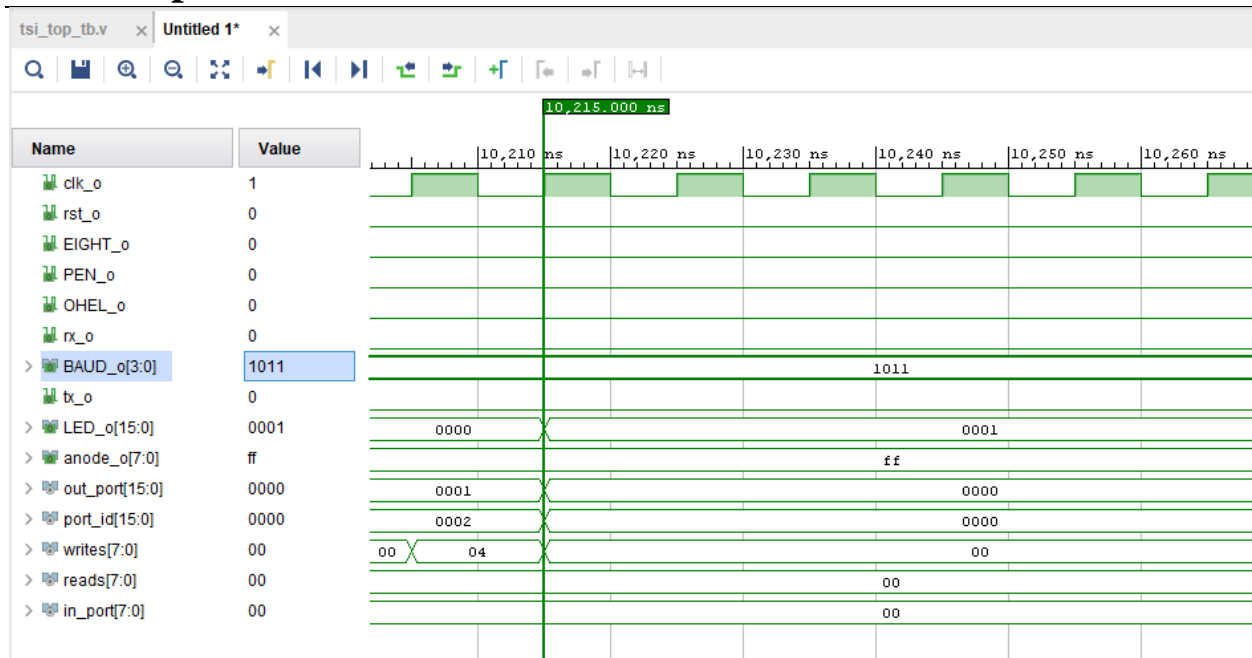


Two cases of the port_id was tried with the read_strobe being active. We see reads 0 bit going active when we read port_id 0000 and bit 1 goes active when reading port 0001.



UART with TSI Chip Specification

7 Chip Level Verification

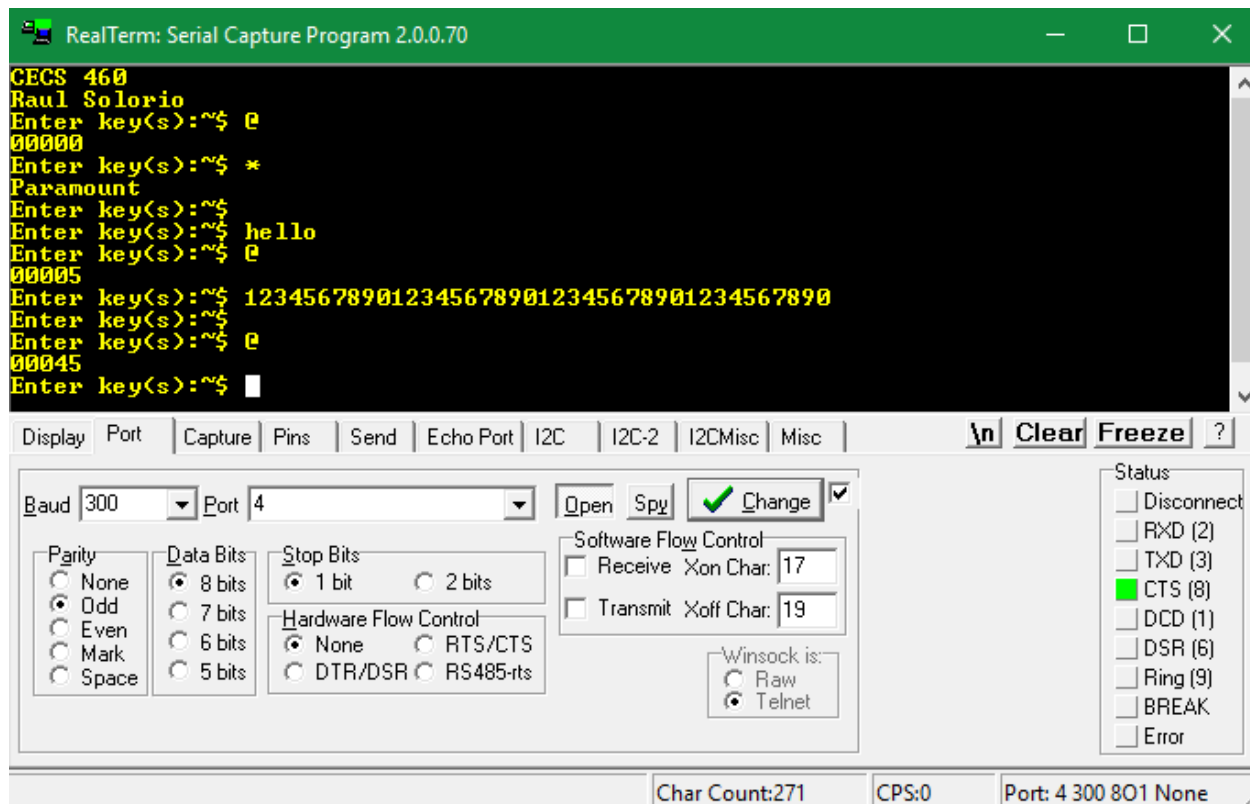


The tsi top level was verified at an instance in time. The signals coming out of the buffers followed the inputs and outputs signal. The tramblaze is outputting the write signal to port 0002 where it sends the value of 0001 to the LED's. This is enough evidence to prove the design is working since the other modules were already verified.



UART with TSI Chip Specification

8 Chip Level Test



The above picture shows how the requirements were implemented on the terminal. Initially the Banner “CECS 460 Raul Solorio” appeared followed by the prompt. After entering the key ‘@’ it displayed the number of characters entered since last reset. The character ‘*’ outputted my city which is Paramount. The enter key displays another prompt and waits for the user to input a character. Each line may contain up to 40 characters after the prompt before it automatically displays a new prompt.

Appendix