CECS 460 – Full UART with TSI

Marc Dominic Cabote

014938597

Spring 2018

FULL UART w/ TSI

Prepared by:	Date:	Revision:
Marc Dominic Cabote	15 May 2018	3

TABLE OF CONTENTS

FULL UART w/ TSI

Prepared by:	Date:	Revision:
Marc Dominic Cabote	15 May 2018	3

TABLE OF TABLES

TABLE OF FIGURES

FULL UART w/ TSI

Prepared by:	Date:	Revision:
Marc Dominic Cabote	15 May 2018	3

1 INTRODUCTION

This project is intended for introducing Technology Specific Instantiations(TSI) with Universal Asynchronous Receiver Transmitter (UART)—with both the transmit and the receive part. At the heart of the Tx and Rx engine are shift registers that shifts in and out data one bit at a time. The baud rate determines the speed at which data will be transferred and received.

1.1 PURPOSE

The purpose of this document is to specify the requirements for the TSI with full UART engine—with both the receive and transmit engine. This document will serve as specifications for the UART. This document will then be modified if new modules will be added to the design.

2 REQUIREMENTS

The requirements for this project are pretty straightforward:

i) Walk the LEDS while idling in the main loop
ii) Display a banner starting out of reset then provide a prompt for the user
iii) When a key is pressed, it will then display what key was pressed on the terminal
iv) Process keys according to the following:
    a. Carriage Return – Send the cursor to start a new line with the new prompt
    b. Backspace – Erase the character in front of the cursor with a blank
    c. Asterisk – will output your hometown followed by a newline and the prompt
    d. At sign – will result in outputting the number of characters received—should counter should restart
v) It should only display up to 40 characters—if it exceeds 40, output to a new line resetting the prompt and the character counter.
vi) Add the TSI block with the UART design

3 TOP LEVEL DESIGN

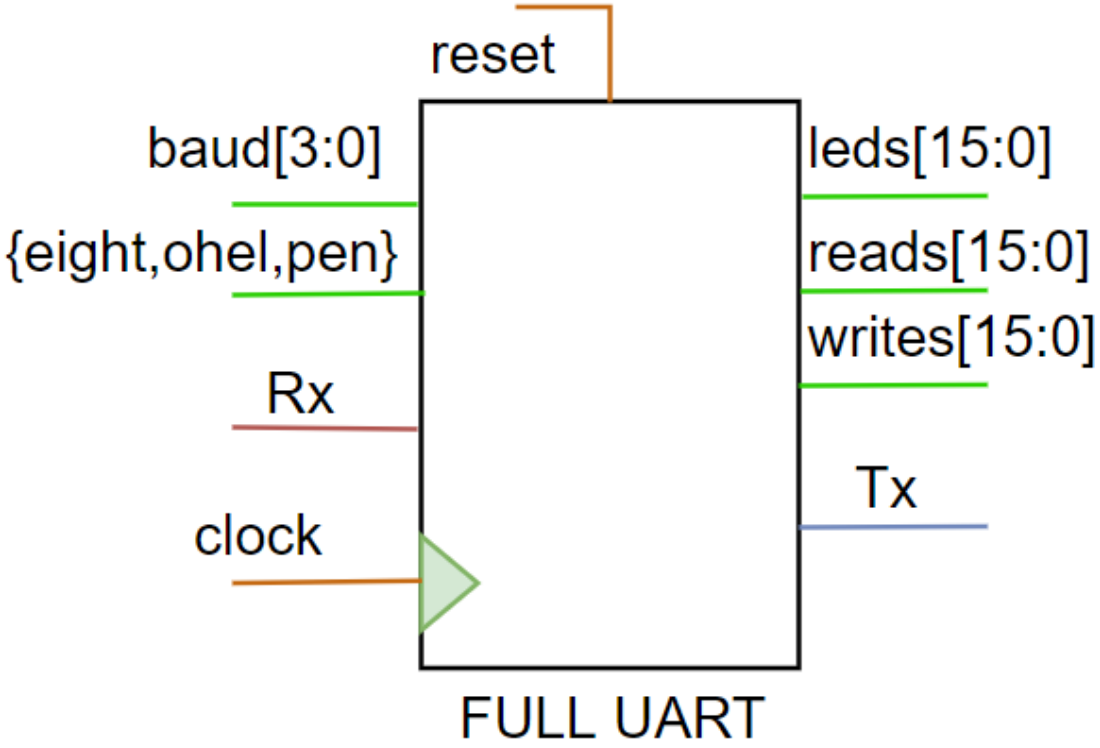


FIGURE 1 TOP LEVEL DESIGN

3.1 DESCRIPTION

The UART engine has the logic needed to communicate with a computer using both the transmit and receive engine. For the terminal to have a stable connection in order to output the proper characters, the program—Realterm—should know what the setting is for the transmit engine—it should know the baud rate, odd or even parity and number of bits. If the program settings match the FPGA settings, it will then be able to the requirements mentioned. If the settings do not match different characters will show up.

The reads and the writes are enables, coming out of the TramelBlaze. The TramelBlaze uses these writes and enables to have specific ports to access data from a variety of sources within the SOC.

3.2 BLOCK DIAGRAM

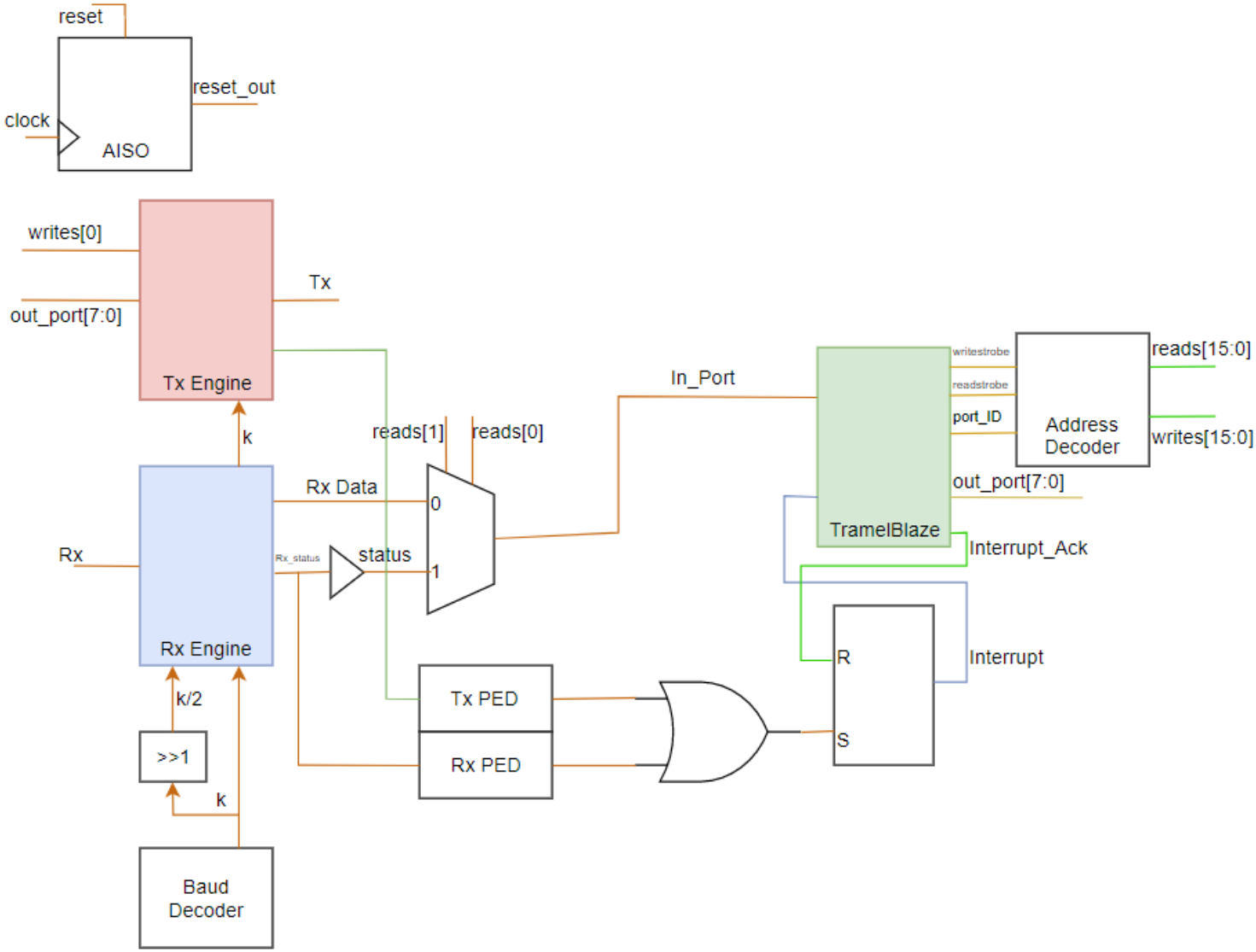


FIGURE 2 TOP LEVEL BLOCK

The AISO synchronizes the reset for the rest of the modules

3.3 I/O

Inputs:

i) 4-bit Baud Rate

TABLE 1

CASE	COUNT	BITS PER SECOND
0000	333,333	300
0001	83,333	1200
0010	41,667	2400
0011	20,833	4800
0100	10,417	9600
0101	5,208	19200
0110	2,604	38400
0111	1,736	57600
1000	868	115200
1001	434	230400
1010	217	460800
1011	109	921600

ii) Eight

-If set high, 8 bits of data will be used instead of default 7

iii) Pen

-Enables parity

iv) OHEL

-If high checks odd parity, if low checks even parity

v) Clock

-100 MHz on-board clock

vi) Reset

-Reset is fed to the AISO which synchronizes the reset for all the registers

vii) Rx

-receives the data

Outputs:

i) LEDS

-for debugging the TramelBlaze

ii) Tx

-transmits the data

3.3.1 Signal Names

Inputs:
Clk
Rst
Eight
Pen
Ohel
Rx
[3:0] baud

Outputs:
Tx
[15:0] leds
[15:0] reads
[15:0] writes

3.3.2 Pin Assignments

TABLE 2 – Pin Assignments

Signal Name	Pin Location	Signal Name	Pin Location
clk	E3	**leds[4]**	R18
ohel	L16	**leds[5]**	V17
pen	M13	**leds[6]**	U17
eight	R15	**leds[7]**	U16
baud[0]	R17	**leds[8]**	V16
baud[1]	T18	**leds[9]**	T15
baud[2]	U18	**leds[10]**	U14
baud[3]	R13	**leds[11]**	T16
rst	N17	**leds[12]**	V15
Tx	D4	**leds[13]**	V14
Rx	C4	**leds[14]**	V12
leds[0]	H17	**leds[15]**	V11
leds[1]	K15	**----------**	**----------**
leds[2]	J13	**----------**	**----------**
leds[3]	N14	**----------**	**----------**

4.1 BLOCK DIAGRAM OF THE TRANSMIT ENGINE

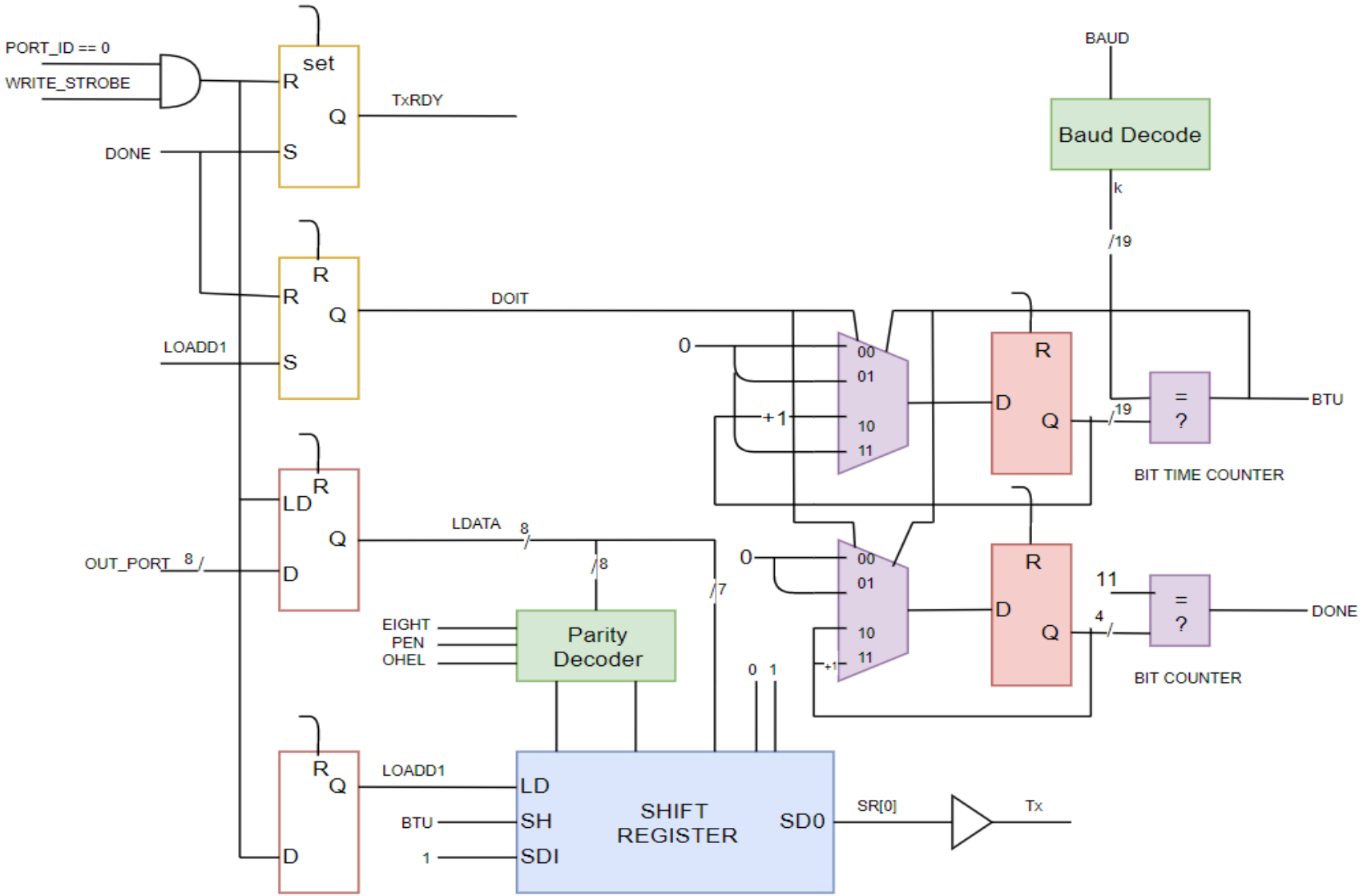


FIGURE 3 DETAILED TRANSMIT ENGINE

The highlighted parts are the important blocks for the transmit engine

4.1.1 Shift Register

The Shift Register with the parity decoder is basically the heart of the transmit engine. The Shift Register shifts out data one bit at a time, depending on the parity decoder settings. The shift register is 11'b11_1111111_11 at default—this tells us that this is the mark, and it is ready to transmit data. Once all the bits of data are shifted out, done and txrdy will go high. This is then recognized by the tramelblaze which converts the data into hex values to be converted to ASCII.

4.1.2 Bit Time Counter

The bit time counter tells us how fast the shift register should shift out data—which we can then say that it determines how fast data should be transmitted. The bit time counter is tied with the baud decoder. This determines when bit time up will go high. Bit time up then goes to the shift register—when set high, it tells the shift register to shift data out.

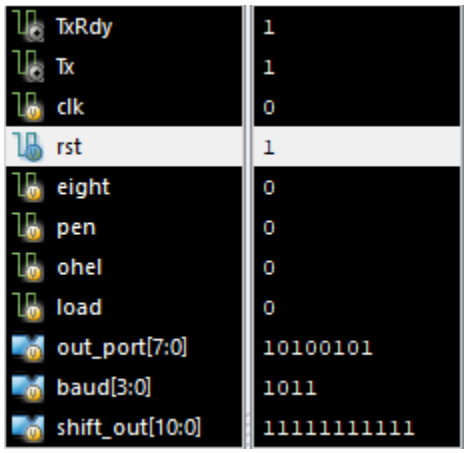
4.1.3 Bit Counter

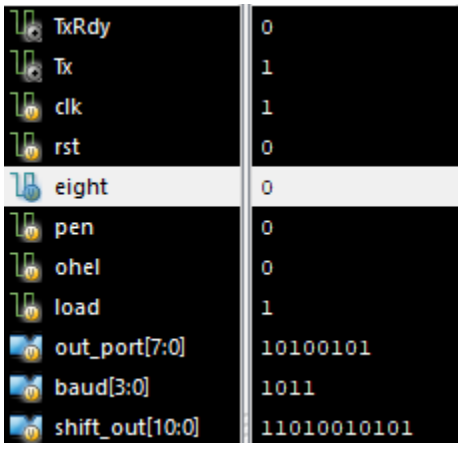
The bit counter has a very similar code with the bit time counter. While the bit time counter takes care of the how fast each data should be transmitted—baud rate—the bit counter counts up to 11 in decimal. It will tell us that once all 12 bits of data have been shifted out, "done" will go high telling us that the data has been transmitted.

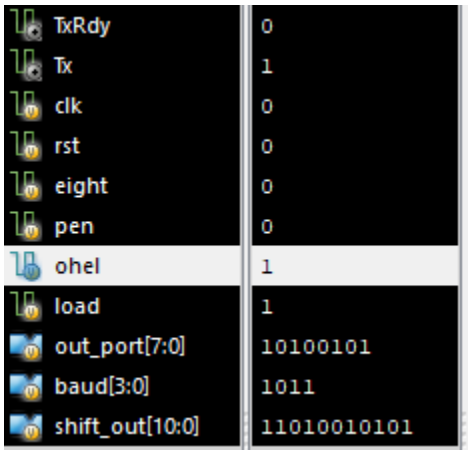
4.2 TRANSMIT ENGINE VERIFICATION

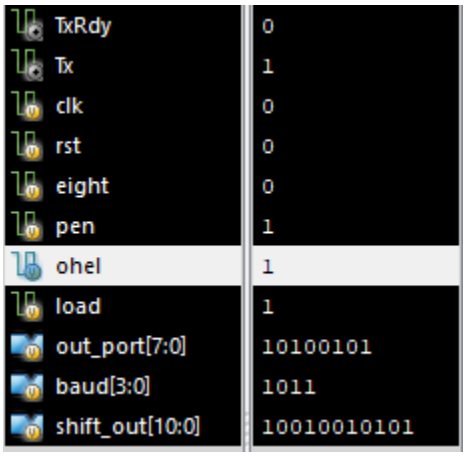
The transmit verification is pretty straightforward out_port is set to be 8'b1010_0101 or 8'hA5. Reset is exercised initially and baud rate is set to the fastest setting for easier verification. All the cases for {eight, pen, ohel} are then verified if they are properly working. Expected outputs are commented in the test fixture to verify shift register outputs on the waveforms. If the expected outputs are the same as the waveform outputs. This then verifies that the transmit engine works. It will also be seen that the TxRdy is high at reset.

4.2.1 Transmit Engine Verification Waveform









These are just some of the verified results. The entire waveform will be shown in the next page

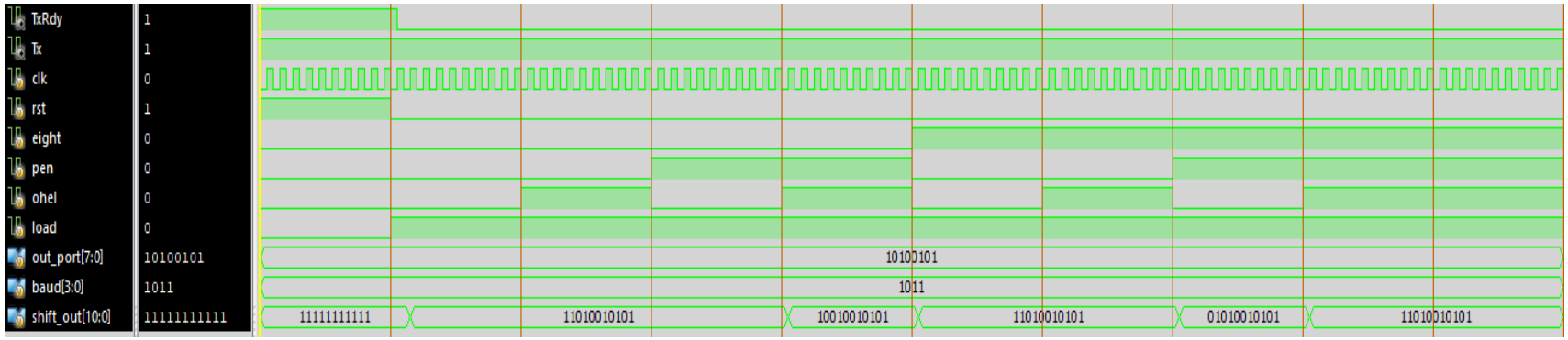


FIGURE 4 TRANSMIT ENGINE VERIFICATION WAVE

5.1 BLOCK DIAGRAM OF THE RECEIVE ENGINE DATA PATH

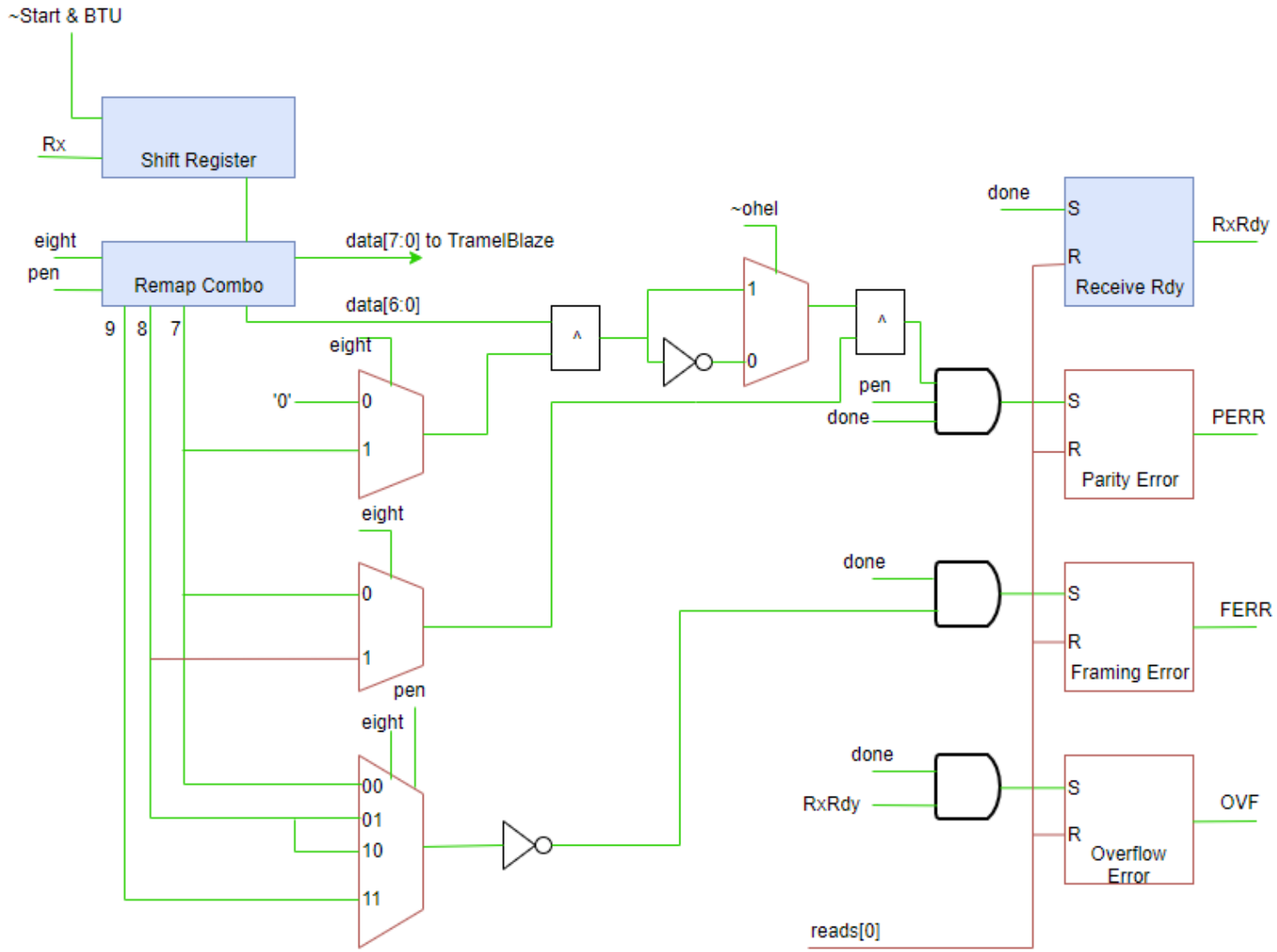


FIGURE 5 RECEIVE ENGINE DATA PATH

The highlighted parts are the important blocks for the receive engine

5.1.1 Shift Register

The shift register inside the receive engine outputs the data to the remap combo. Since the Rx input will be the opposite of the Tx Output. The data received has to go through the remap combo to match the data transmitted.

5.1.1 Remap Combo

The remap combo shifts out data to the right depending on the configuration of {eight, pen}. It ensures that the data is held in [7:0] that is then sent out to the TramelBlaze. If select is 7N1, then data is shifted twice to the right so it would be in [7:0]. If select is 7P1 or 8N1, data is shifted only once to be in [7:0]. And if select is 8P1, the data stays the same.

5.2 BLOCK DIAGRAM OF THE RECEIVE ENGINE CONTROL

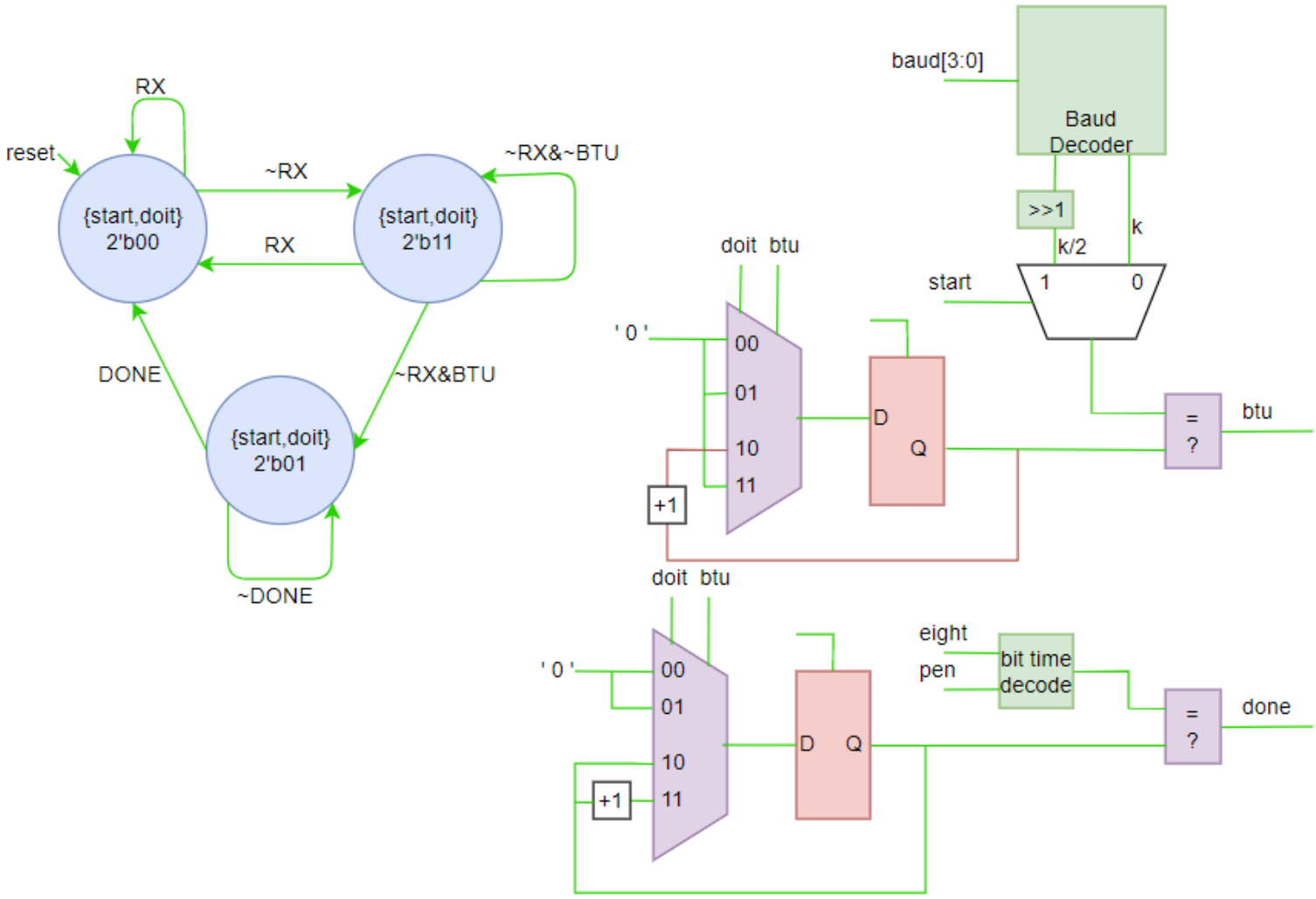


FIGURE 6 RECEIVE ENGINE CONTROL

The highlighted parts are the important blocks for the receive engine

5.2.1 State Machine

The state machine is the heart of the receive engine—it controls the data path. It waits for the start bit, then it will wait half the bit time to make sure that the data received is the data transmitted.

5.2.2 Bit Time Counter Divider

The bit time divider goes to the state machine to dived the bit time in half. This will then go to the state machine to ensure that the data is being sampled properly.

5.2.3 Bit Time Counter

Just like in the transmit engine, the bit time counter tells us how fast the shift register should shift data—however this time, it is shifting in data instead of out. It will start being divided by two to ensure that the data being received is the data being transmitted. Bit time up then goes high which tells the state machine when to switch states.

5.2.4 Bit Counter

Just like in the transmit engine, the bit counter has a similar functionality with the bit time counter. This time, {eight, pen} is taken into account. The receive end has to know how many bits of data is being transferred. This will then hold the data on the wire for a certain amount of time. When it is done counting the DONE bit will then go high, which then goes to the state machine to finish the sequence

5.3 RECEIVE ENGINE VERIFICATION

5.3.1 Receive Engine Verification Waveform

The receive engine’s verification is pretty straightforward. Rx was set high and low to produce mimic receiving 0xA5. However, since it is the receive end if it was not 0xA5, the data will be reversed. I chose 0xA5 since if the receive end works properly it should get 0xA5 as well.

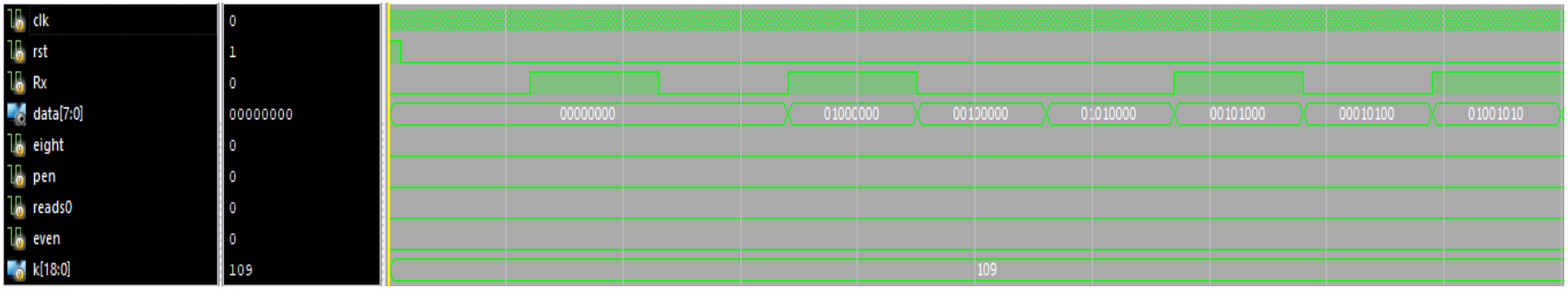


FIGURE 7 RECEIVE ENGINEVERIFICATION WAVE

6 TECHNOLOGY SPECIFIC INSTANTIATIONS

Technology Specific Instantiations (TSI) contains all the references to the target technology which is the full UART. In this case, it is specific to our UART, and will only work with it. It is one big buffer for all the signals in our UART design so it would meet the timing requirements of our design.

5.3 BLOCK DIAGRAM OF UART WITH TSI

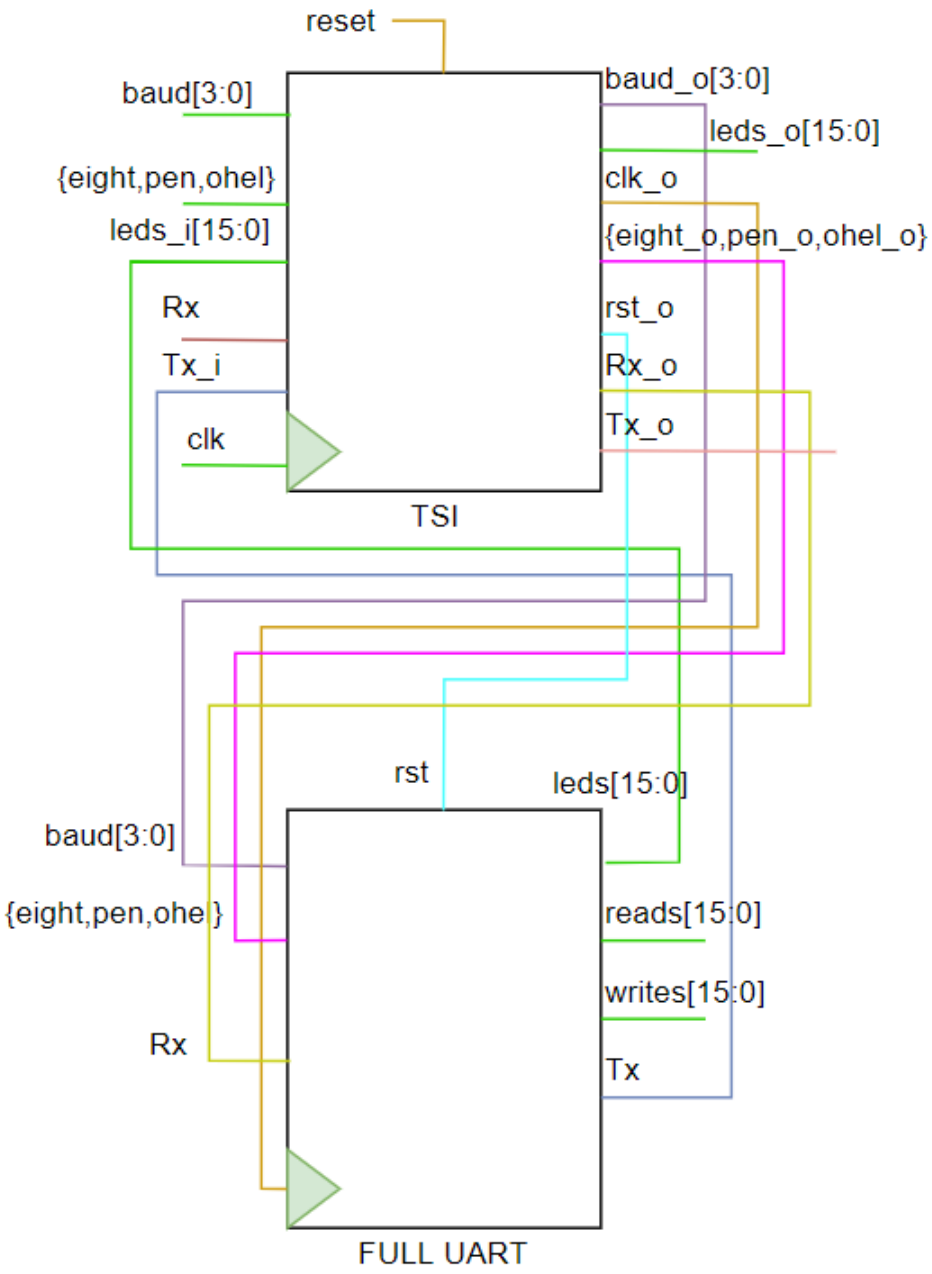


FIGURE 8 FULL UART w/ TSI

7 THE TRAMELBLAZE

The Tramelblaze's purpose for this project is an external block which is used to output the hex values needed. The hex values will then be converted by Realterm as ASCII characters. This will then display the desired output depending on what the user wants. It also outputs values at a different port which is tied to the LEDs to for debugging purposes.

A SOURCE CODES

The codes will all be presented starting in the next page onwards