TrameBlaze™

Full UART Chip Specification

CECS 460: System on Chip Design

Submitted by:

Bryan Linares

008236252

TramelBlaze™

Prepared by:	Loc/Dept	Date:	Document Filename	Revision:
B. Linares	CECS 460	May 15, 2018	Final Specification	4.0

Table of Contents

1 Introduction

This design Specification is for a Full UART design targeting and tested on the Artix 7, Nexys 4 DDR FPGA platform.

This Chip Specification intends to provide a detailed description of the Full UART System on Chip created using the TramelBlaze processor with the platform target of the Nexys 4 FPGA. The UART is capable of serial input and output at varying time rates and was created incrementally. First draft revisions involved only implementing and testing the TramelBlaze processor. The second version was a design and implementation of the Transmit Engine portion of the UART, using the TramelBlaze for as a data store and memory output device. The next revision involved implementing the Receive engine portion of the UART, which involves properly capturing and interpreting input ascii character data from a computer terminal. This fourth and final revision adds the Technology Specific Instantiation block for further control of the electronics of the chip design.

1.1 Purpose

The Purpose of this document is to specify the requirements of this Full UART for proper use and implementation when needed. The document will elaborate on the design's top level specifications and every module contained therein along with design methods and methodology explanations with the goal of clarification. It will also cover the software created for use within the TramelBlaze instantiation.

2 Applicable Documents

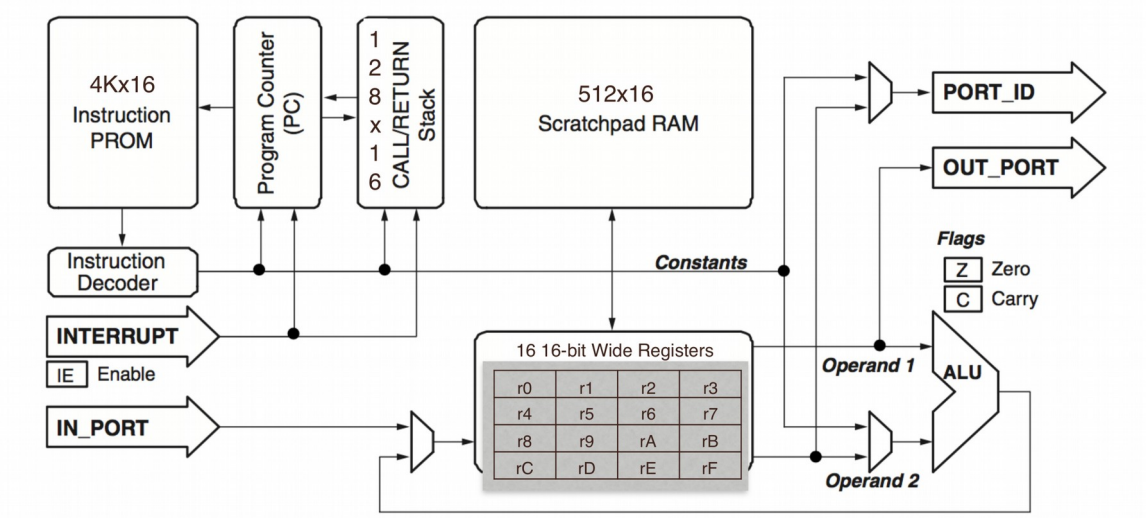
2.1 External Documents

Information for used technologies not completely covered within this document.

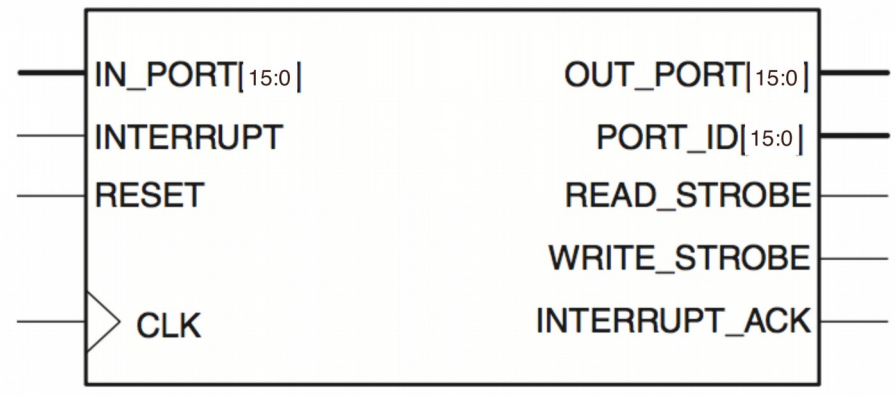
2.1.1 PicoBlaze

The TramelBlaze is a 16-bit emulator of the 8-bit PicoBlaze processor. The PicoBlaze Embedded Processor by Xilinx cannot be used on the Nexys 4 target platform for this design. As such, the TramelBlaze was created by John Tramel as a 16-bit emulator to allow the use of this embedded micro-controller in this SoC design. The TramelBlaze implements the entirety of the Instruction Set Architecture of the PicoBlaze and so instruction references can be used from the original with a few key differences in syntax limited by the custom assembler used for this design. The only

TramelBlaze Architecture: note the 16-bit width on the memories.



The Top module for the TramelBlaze: Identical to PicoBlaze except for the 16bit data widths.

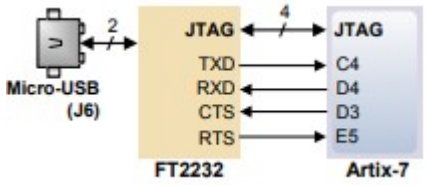


0	NOP	29	RETURN
1	ADD rX, kk	30	RETURNC
2	ADD rX, rY	31	RETURNNC
3	ADDCY rx, kk	32	RETURNZ
4	ADDCY rX, rY	33	RETURNNZ
5	AND rX, kk	34	RETDIS
6	AND rX, rY	35	RETEN
7	CALL aaa	36	RL rX
8	CALLC aaa	37	RR rX
9	CALLNC aaa	38	SL0 rX
10	CALLZ aaa	39	SL1 rX
11	CALLNZ aaa	40	SLA rX
12	COMP rX, kk	41	SLX rX
13	COMP rX, rY	42	SR0 rX
14	DISINT	43	SR1 rX
15	ENINT	44	SRA rX
16	INPUT rX, (rY)	45	SRX rX
17	INPUT rx, pp	46	SUB rX, kk
18	JUMP aaa	47	SUB rX, rY
19	JUMPC aaa	48	SUBC rX, kk
20	JUMPNC aaa	49	SUBC rX, rY
21	JUMPZ aaa	50	TEST rX, kk
22	JUMPNZ aaa	51	TEST rX, rY
23	LOAD rX, kk	52	XOR rX, kk
24	LOAD rX, rY	53	XOR rX, rY
25	OR rX, kk	54	FETCH rX, kk
26	OR rX, rY	55	FETCH rX, (rY)
27	OUTPUT rX, (rY)	56	STORE rX, kk
28	OUTPUT rX, pp	57	STORE rX, (rY)

Above is the list of available instructions to the TramelBlaze. Input and Output operations work identically to the PicoBlaze.

2.1.2 Nexys4 DDR

The Nexys4 DDR is the development board used for this design in implementation, testing and verification. It features Xilinx's Artix 7 field programmable gate array and offers a high capacity of logic and resources. Interfacing with the board in this design is limited to general purpose switches and buttons as well as LEDs. Of crucial use is the UART communication port which lies within its shared UART/JTAG USB micro connection.



The USB JTAG and UART functions exist at the above connections on the Nexys4 board.

Both functions behave entirely independently from one another. The circuitry does not interfere with the other and so both can be used freely. The UART circuitry allows the two wire serial ports (TXD and RXD above) which are used to receive and send data in our design. The RTS and CTS ports are optional hardware flow control signals and are not used here.

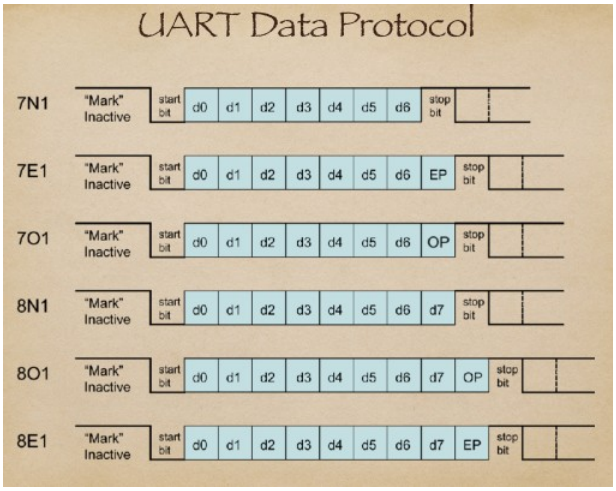
Drivers, software or terminals on the PC must direct data to the COM port to produce serial data traffic that will be seen on the C4 and D4 pins of the FPGA.

2.1.3 Industry standard RS-232

The UART protocol allows communication across two devices without including a clock across the interface. All that this RS232 protocol requires on both sides of the communication is the baud rate, number of bits being sent per transmission, and parity expectation.

The RS232 standard specifically defines some signal characteristics such as voltage levels, timing, pin identifications, and to standards of functions of the circuits withing the interface connections among other elements. The majority of these electronic concerns are set by the tools used.

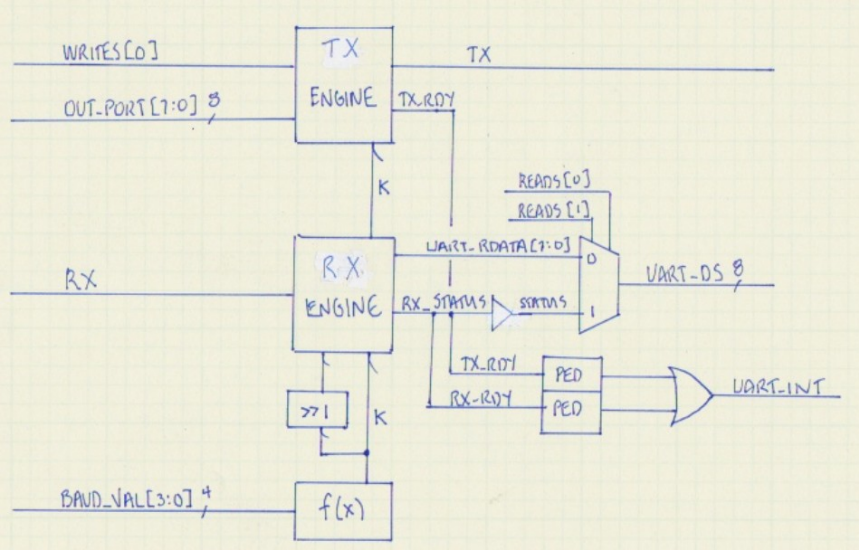
The concerns within this design are the proper packaging and reception of data to the standard most specifically set by Actel in their CoreUART.



2.2 Internal Documents

2.2.1 Internally generated specifications

Technical specifications remained within the set standards. The following top level block specification served as a guide to this design's HDL methodologies: the interrupts, ports, reads, and writes signals interface with the TramelBlaze in order to properly communicate with the terminal along the TX and RX lines. Lines are organized and blocks are designed as simplistically as possible for ease of implementation.



2.2.2 Applicable methodology documents

Adapted from Henessy/Patterson the methodology used in this digital design is presented within two stages, the data path, and control logic. Data must first flow and all registers, counters, data selections, and functional clouds must be defined. State machines are created in a Modified Moore fashion and as such separate combinational and synchronous sections for ease of implementation and readability.

2.2.3 Chip test plan

The design is to be developed and tested incrementally. The transmit engine portion is created and tested first, and runs autonomously, sending data readable at the terminal. Then the Receive engine is created, to capture data properly and send it out using the previous transmit engine as proof of proper operation. Finally, this revision adds the Technology Specific Instantiation block for more control over the I/O ring and rates of the design.

3 Requirements

3.1.1 Interface Requirements

The design incorporates the TramelBlaze and UART components within a Core module going through the TSI to communicate externally. All switches and buttons and LEDs from physical Inputs and Outputs from the board go through the TSI before interacting with the design within the FPGA. As the previous specification shows the data is captured by the receive engine first and sent into the TramelBlaze, which is interrupted and acts according to the software's reaction to the received data. The transmit engine takes in data output by the microcontroller and outputs it to the proper time as specified by the industry standard. The two UART engine modules do not communicate directly to each other.

The interface can be adjusted by the baud rate, and parity bit specification, and LEDs are used only to show that the design is continuously operating.

When errors are encountered in receiving data, the design is also capable of detecting:

Overflow: Receive ready signal is still asserted beyond expected amount of bits.
Framing Errors: The mark and stop bits are not received properly to denote end of com.
Parity Errors: When the recalculated parity does not match the one received.

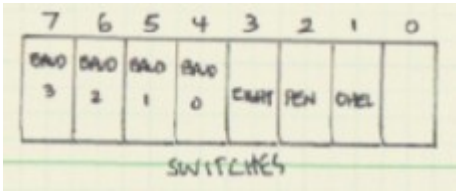
Transmit and Receive ready signals are used to interrupt and interface with the TramelBlaze.

3.1.2 Physical Requirements

The most important requirement among timing is the Baud Rate, controlled by slide switches 4-7 on the Nexys 4 DDR. This is the amount of clocks counted assuming 100mhz on the Nexys4, that each bit is to be held on the line.

baud[3:0]	rate	bit time	Eng Not	N2 Count	N3/N4 Count	N2 bits	N3/N4 bits
0000	300	0.003333333	3.3333 ms	166,667	333,333	18	19
0001	1200	0.000833333	833.33 us	41,667	83,333		
0010	2400	0.000416667	416.66 us	20,833	41,667		
0011	4800	0.000208333	208.33 us	10,417	20,833		
0100	9600	0.000104167	104.16 us	5,208	10,417		
0101	19200	5.20833E-05	52.083 us	2,604	5,208		
0110	38400	2.60417E-05	26.041 us	1,302	2,604		
0111	57600	1.73611E-05	17.361 us	868	1,736		
1000	115200	8.68056E-06	8.6806 us	434	868		
1001	230400	4.34028E-06	4.3403 us	217	434		
1010	460800	2.17014E-06	2.1701 us	109	217		
1011	921600	1.08507E-06	1.0851 us	54	109		
	N2: Nexys 2						
	N3: Nexys 3						
	N4: Nexys 4						

Switches 1-3 control the size of the data sent (seven or eight bits), switch 2 is parity enable, which turns the parity bit on or off, and switch 1 denotes Odd parity when High, Even when Low.



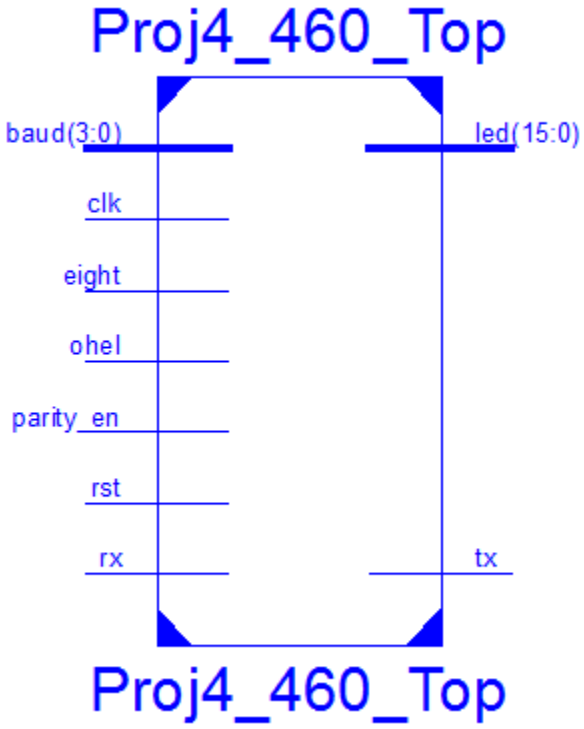
LEDs are flashed to denote operation, and Center Button on the Nexys 4 is used for high active RESET.

4 Top Level Implementation

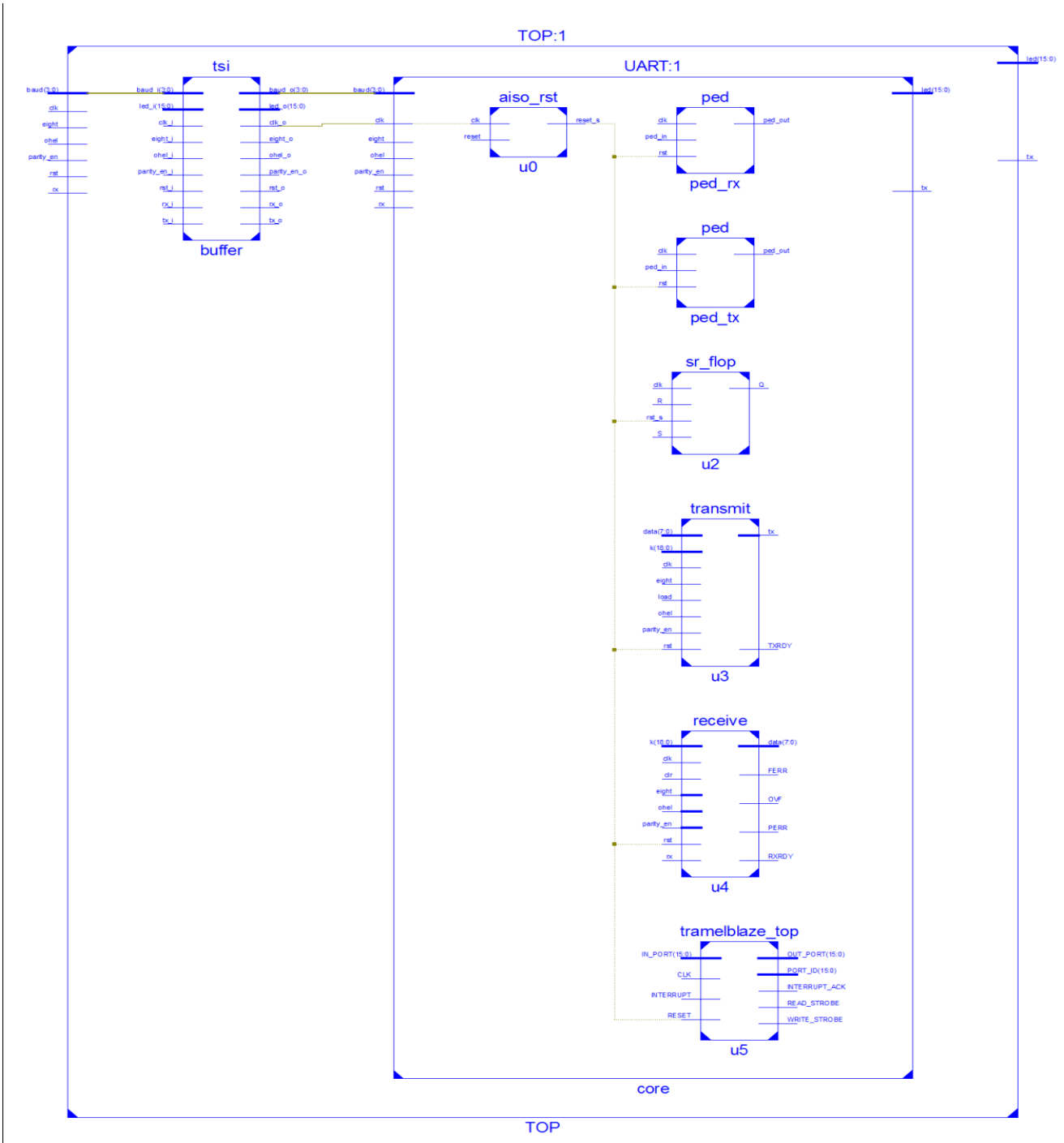
4.1 Design Description

The top level module is responsible for interconnecting the Core block and the TSI. The TSI contains all the reference to target technology libraries specific to our design and sets certain options available to each. All I/O communications go through the TSI block.

4.2 Block Diagram



4.2.1 Detailed Top Level Block Diagram



4.3 Data Flow Description

The sign is controlled by the single 100mhz clock from the board. All Input and Output data flows through the TSI block first, then into the Core UART design. Input from switches are decoded in the UART and are used within the two directional engines. Input data flows from the board interfaces and into the UART block. The serial terminal sends data into the design which goes into the Receive engine, which then goes into the TramelBlaze, which passes it to the Transmit engine after processing it. The tramelblaze also flashes the LEDs to show that it is working, this is controlled by a busy-wait timer within the software for visiblity. Since the TramelBlaze processor serves as an intermediary for all operations, the address decode block is used to simplify the design of the read and write signals for interfacing among different sections of the block. The addition of the TSI block does not affect interactions with the device.

4.4 Input/Output Interface Description

4.4.1 Signal Names

Signal(In)	Description
Clk	Clock signal from board
rst	Reset signal from button
eight	Eight length data when high
parity_en	Enable parity bit
ohel	Odd parity high, Even Parity set low
rx	Receive data pin
Baud(3:0)	Baud rate choose on switches
Signal(O)	Description
LED(15:0)	LED array output
TX	Transmit data pin

4.4.2 Pin List

```
## Clock signal
NET "clk"               LOC = "E3"| IOSTANDARD = "LVCMOS33";

## Switches
NET "ohel"              LOC=L16 | IOSTANDARD=LVCMOS33; #IO_L3N_T0_DQS_EMCCLK_14
NET "parity_en"         LOC=M13 | IOSTANDARD=LVCMOS33; #IO_L6N_T0_D08_VREF_14
NET "eight"             LOC=R15 | IOSTANDARD=LVCMOS33; #IO_L13N_T2_MRCC_14
NET "baud[0]"           LOC=R17 | IOSTANDARD=LVCMOS33; #IO_L12N_T1_MRCC_14
NET "baud[1]"           LOC=T18 | IOSTANDARD=LVCMOS33; #IO_L7N_T1_D10_14
NET "baud[2]"           LOC=U18 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A13_D29_14
NET "baud[3]"           LOC=R13 | IOSTANDARD=LVCMOS33; #IO_L5N_T0_D07_14

## Buttons
NET "rst"              LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14

## LEDs
NET "led[0]"          LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "led[1]"          LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "led[2]"          LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
NET "led[3]"          LOC=N14 | IOSTANDARD=LVCMOS33; #IO_L8P_T1_D11_14
NET "led[4]"          LOC=R18 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_D09_14
NET "led[5]"          LOC=V17 | IOSTANDARD=LVCMOS33; #IO_L18N_T2_A11_D27_14
NET "led[6]"          LOC=U17 | IOSTANDARD=LVCMOS33; #IO_L17P_T2_A14_D30_14
NET "led[7]"          LOC=U16 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A12_D28_14
NET "led[8]"          LOC=V16 | IOSTANDARD=LVCMOS33; #IO_L16N_T2_A15_D31_14
NET "led[9]"          LOC=T15 | IOSTANDARD=LVCMOS33; #IO_L14N_T2_SRCC_14
NET "led[10]"         LOC=U14 | IOSTANDARD=LVCMOS33; #IO_L22P_T3_A05_D21_14
NET "led[11]"         LOC=T16 | IOSTANDARD=LVCMOS33; #IO_L15N_T2_DQS_DOUT_CSO_B_14
NET "led[12]"         LOC=V15 | IOSTANDARD=LVCMOS33; #IO_L16P_T2_CSI_B_14
NET "led[13]"         LOC=V14 | IOSTANDARD=LVCMOS33; #IO_L22N_T3_A04_D20_14
NET "led[14]"         LOC=V12 | IOSTANDARD=LVCMOS33; #IO_L20N_T3_A07_D23_14
NET "led[15]"         LOC=V11 | IOSTANDARD=LVCMOS33; #IO_L21N_T3_DQS_A06_D22_14

##USB-RS232 Interface
NET "tx"               LOC=D4 | IOSTANDARD=LVCMOS33; #IO_L11N_T1_SRCC_35
NET "rx"               LOC=C4 | IOSTANDARD=LVCMOS33; #IO_L7P_T1_AD6P_35
```

4.4.3 Electrical Characteristics

Standard electrical expectations are shown in the list above for the Nexys 4.

4.5 Clocks

The only clock within the design is the 100MHz clock sourced from the main oscillator of the Nexys 4 DDR board.

4.6 Resets

One reset signal is provided to every synchronous block within the design, set to the Center Button on the Nexys 4, and is HIGH active.

4.7 Software Requirements

The software created for this design has numerous requirements to test proper reception and packaging of data:

- The LEDs are rotating a 1 left as the software idle in the main loop.
- A banner is displayed when starting out of reset. Then a prompt is printed.

- When a key is pressed a byte of data is sent to the RX, notified in the RX ready signal. The signal is read and the appropriate branch is taken depending on the source of the interrupt, which can be the Transmit, Receive engine or both.

  When receiving the following ASCII characters a different action will be taken:

- <CR> Carriage return: cursor moved to new line, prompt refreshed
- <BS> Backspace: the character beside the cursor is erased
- '*' Asterisk: The designer's hometown and a new line is printed, here it is Los Angeles
- '@' At Symbol: The count of characters on the line so far is printed
- All other characters are echoed on a line up until the 40th one, when a newline prompt is printed.

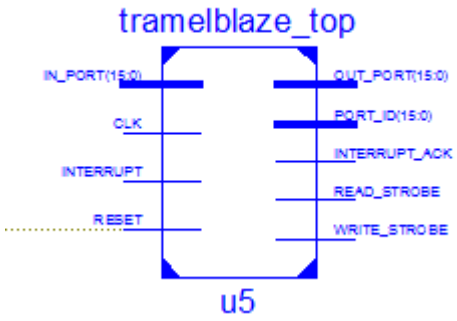
5 Externally Acquired Blocks

5.1 TramelBlaze

5.1.1 Description

A 16-bit microcontrollers instantied within our design that emulates the PicoBlaze. The software is assembled and loaded as a ROM, which will instruct the processor to perform our required actions. In this design the tramelblaze receives ASCII bytes after being captured by the receive engine and gives bytes to the transmit engine to output based on the received characters.

5.1.1 Block Diagram



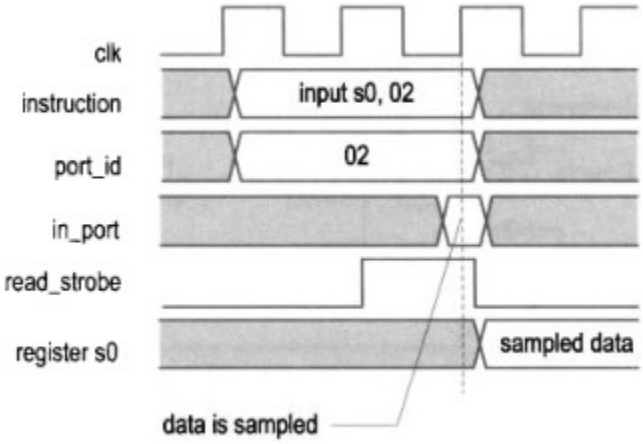
5.1.2 I/O

Signal(In)	Description
IN_PORT	Arbitrary Port ID for input decoding
CLK	Clock Signal
INTERR.	Rising edge signals interrupt, calls ISR
RESET	Reset signal

Signal(O)	Description
OUT_PORT	LED array output
PORT_ID	Output port ID for output decoding
INTERRUPT_ACK	To acknowledge interrupt prepared for another
READ_STROBE	Goes high when reading data from INPUT
WRITE_STROBE	Goes high when writing data from OUTPUT

5.1.3 State Machine Description

TramelBlaze is a straightforward 16-bit microcontroller, which constantly fetches and executes the instructions programmed to it. Its input and output interface is essential to communicating with it.



An input instruction form.

5.1.1 Register Map

The TramelBlaze contains 16 Registers R0...RF in its PROM. All reset to zero and available for general purpose use.

6 Internally Developed Blocks

6.1 UART

In this design, the Core block is the UART and receives and outputs all signals of the design.

6.1.1 Receive Engine

The receive engine captures input data from serial terminal and passes it to the TramelBlaze.

6.1.1 Transmit Engine

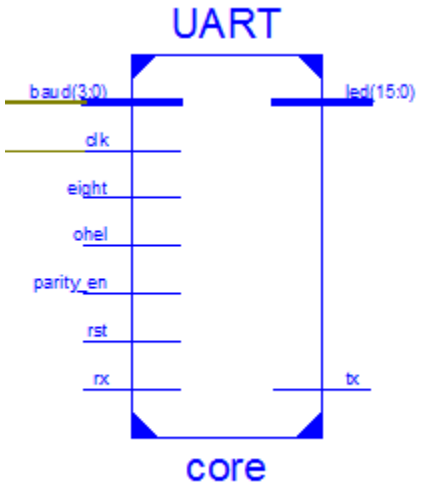
The transmit engine output ASCII bytes with the proper timing set by the switches.

Signal(In)	Description
Clk	Clock signal from board
rst	Reset signal from button
eight	Eight length data when high
parity_en	Enable parity bit
ohel	Odd parity high, Even Parity set low
rx	Receive data pin
Baud(3:0)	Baud rate choose on switches
Signal(O)	Description
LED(15:0)	LED array output
TX	Transmit data pin

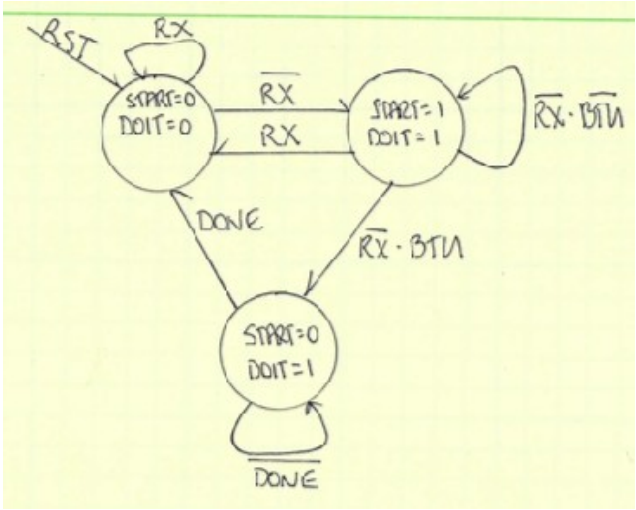
6.1.2 Performance Requirements

The UART assumes a 100MHz clock input for various timing related operations. Crucially, the baud rate is dependent on clock counting from this assumption.

6.1.3 Block Diagram



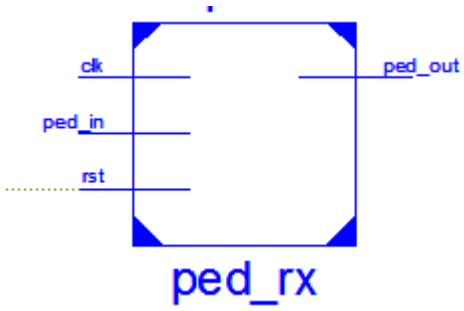
6.1.4 State Machine Descriptions



The Receive engine state machine constantly polls for the rx signal and begins its operation when it gets it.

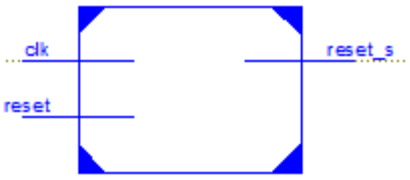
6.2 Positive Edge Detect(PED)

The positive edge detect module is used in numerous places to ensure a signal is held long enough to be acknowledge by the TramelBlaze. It pulses a clock-wide high signal when it receives an input.



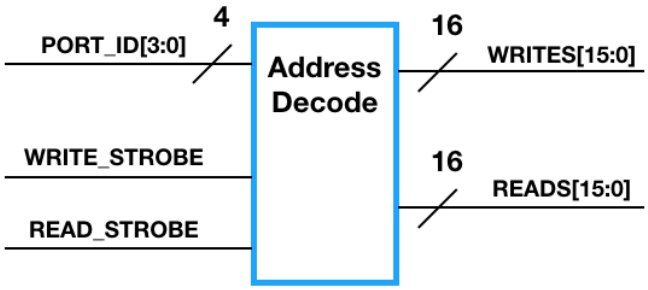
6.3 Asynchronous In Synchronous Out (AISO) for Reset

The AISO prevents flops from entering a metastable state when receiving a reset signal which comes from a timing domain external to the rest of the design.



6.4 Address Decoder

The Address Decoder provides an interface to reading the read or write signals coming out of the TramelBlaze so that they may be used more efficiently across different blocks in the design.

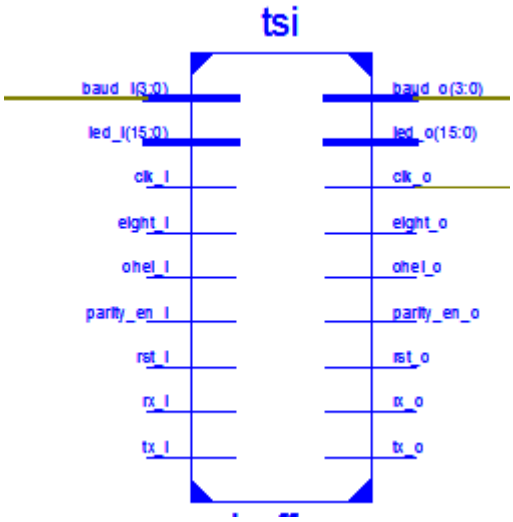


6.5 RSFLop

The RS Flop is a Reset/Set flop, which is set to zero by it's R signal, and 1 by its S.

6.6 TSI

The TSI Block controls slew rates and voltage options for the signals coming in and out of the design. It allows more control over the power usage of the board by the design.



7 Appendices (Source Code)

```
; bryan linares cecs 460 full_uart.tba
        ;
        ;//*************************************************************//
        ;// 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 Spring 2018
        ;// Project name: Project 4 Full UART with TSI
        ;// File name: full_uart.tba
        ;//                                                             //
        ;// Created by Bryan Linares on 5/8/18
        ;// Copyright © 2018 Bryan Linares. All rights reserved.
        ;//                                                             //
        ;// Abstract: TramelBlaze Software for Full UART
        ;// Edit history: 5/15  4.0 Prep for Final Submit
        ;//                                                             //
        ;// 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 sources 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 //
        ;//*************************************************************//
        ASCII_0      EQU   0030
        ASCII_1      EQU   0031
        ASCII_2      EQU   0032
        ASCII_3      EQU   0033
        ASCII_4      EQU   0034
        ASCII_5      EQU   0035
        ASCII_6      EQU   0036
        ASCII_7      EQU   0037
        ASCII_8      EQU   0038
        ASCII_9      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
ASCII_NL      EQU   0000
ASCII_BS      EQU   0008
ASCII_LF      EQU   000A
ASCII_CR      EQU   000D
ASCII_AS      EQU   002A  ; asterisk
ASCII_AT      EQU   0040  ; at
ASCII_COL     EQU   003A  ; colon
ASCII_SP      EQU   0020  ; space
ASCII_DL      EQU   0024  ;dollar sign
TEMP          EQU    R0
POINTER       EQU    R3  ; into scratch
STATUS        EQU    R1  ; UART status  port_id 0000
DATA          EQU    R2  ; UART Data port_id 0001
LEDS          EQU    R4
FLAG          EQU    RC
CHAR_COUNT    EQU    RD
WAITLOWER     EQU    R5 ;for large number for busy wait
WAITUPPER     EQU    R6

;Init all regs and pointers, scratch
      LOAD    TEMP,          0000
      LOAD    POINTER,       0000
      LOAD    CHAR_COUNT,    0000
      LOAD    STATUS,        0000
      LOAD    DATA,          0000
      LOAD    FLAG,          0001
      LOAD    WAITLOWER,     0000
      LOAD    WAITUPPER      0000
      LOAD    LEDS,          0001
      CALL    LOADBANNER
      CALL    LOADPROMPT
      CALL    LOADHOME
      CALL    LOADERASE
      CALL    LOADNEWLINE
      LOAD    POINTER, 0000
      ENINT
      JUMP main

LOADBANNER
      LOAD    TEMP, ASCII_AS
LOOP50  STORE   TEMP, POINTER
      ADD     POINTER, 0001
      ADD     CHAR_COUNT, 0001
      COMP    CHAR_COUNT, 0031
      JUMPC   LOOP50
      LOAD    CHAR_COUNT, 0000

      COMP    POINTER, 0064
      JUMPNC  BANNERNEWLINE

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

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

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

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

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

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

LOAD    TEMP, ASCII_B
STORE   TEMP, POINTER
ADD     POINTER, 0001

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

LOAD    TEMP, ASCII_Y
STORE   TEMP, POINTER
ADD     POINTER, 0001

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

LOAD    TEMP, ASCII_B
STORE   TEMP, POINTER
ADD     POINTER, 0001

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

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

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

LOAD    TEMP, ASCII_C
STORE   TEMP, POINTER
```

```
ADD     POINTER, 0001

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

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

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

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

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

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

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

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

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

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

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

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

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

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

LOAD    TEMP, ASCII_SP
```

```
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

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

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

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

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

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

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

        LOAD    TEMP, ASCII_AS
        JUMP    LOOP50

BANNERNEWLINE

        LOAD    TEMP, ASCII_CR

 STORE   TEMP, POINTER
 ADD     POINTER, 0001
 LOAD    TEMP, ASCII_LF
 STORE   TEMP, POINTER
 ADD     POINTER, 0001
 LOAD    CHAR_COUNT, 0000
 RETURN

LOADPROMPT

 LOAD    TEMP, ASCII_B
 STORE   TEMP, POINTER
 ADD     POINTER, 0001

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

 LOAD    TEMP, ASCII_Y
 STORE   TEMP, POINTER
 ADD     POINTER, 0001

 LOAD    TEMP, ASCII_A
 STORE   TEMP, POINTER
```

```
        ADD     POINTER, 0001

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

        LOAD    TEMP, ASCII_AT
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

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

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

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

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

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

        LOAD    TEMP, ASCII_DL
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

        RETURN

LOADHOME

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

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

        LOAD    TEMP, ASCII_M
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

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

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

        LOAD    TEMP, ASCII_O
```

```
STORE   TEMP, POINTER
ADD     POINTER, 0001

LOAD    TEMP, ASCII_W
STORE   TEMP, POINTER
ADD     POINTER, 0001

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

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

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

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

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

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

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

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

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

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

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

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

LOAD    TEMP, ASCII_S
STORE   TEMP, POINTER
ADD     POINTER, 0001
```

```
        LOAD    TEMP, ASCII_SP
        STORE   TEMP, POINTER
        ADD     POINTER, 0001

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

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

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

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

        RETURN

LOADERASE

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

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

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

        RETURN

LOADNEWLINE

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

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

        RETURN

WAIT
        ADD     WAITLOWER, 0001
        ADDC    WAITUPPER, 0000
        COMP    WAITUPPER, 001F
        JUMPNZ  WAIT
        LOAD    WAITLOWER, 0000
        LOAD    WAITUPPER, 0000
        RETURN
```

```
        ADDRESS 0300
ISR
        INPUT   STATUS, 0001
        AND     STATUS, 0003 ;mask
        COMP    STATUS, 0003
        JUMPZ   GOT_BOTH
        COMP    STATUS, 0002        ; test TXRDY
        CALLZ   GOT_TXRDY
        COMP    STATUS, 0001        ; test RXRDY
        CALLZ   GOT_RXRDY
        RETEN

GOT_BOTH
        CALL GOT_TXRDY
        CALL GOT_RXRDY
        RETEN

; BIN2ASCII: convert hex val to ascii number
BIN2ASCII
        LOAD    RE, CHAR_COUNT

        LOAD    RD, 000A
        CALL    FIND_IT
        ADD     RB, 0030
        STORE   RB, 00C2

        ADD     RE, 0030
        STORE   RE, 00C3

        RETURN

FIND_IT
        LOAD    RB, 0000
SUBTRACT SUB    RE, RD
        JUMPC   RESTORE
        ADD     RB, 0001
        JUMP    SUBTRACT
RESTORE ADD     RE, RD
        RETURN

; Branched routines
GOT_TXRDY
        COMP    FLAG, 0000
        RETURNZ
        FETCH   TEMP, POINTER
        OUTPUT  TEMP, 0000
        ADD     POINTER 0001
        COMP    FLAG, 0001
        JUMPZ   PRINT_BANNER
        COMP    FLAG, 0002
        JUMPZ   PRINT_PROMPT
        COMP    FLAG, 0003
        JUMPZ   PRINT_HOMETOWN
        COMP    FLAG, 0004
        JUMPZ   PRINT_BS
        COMP    FLAG, 0005
        JUMPZ   PRINT_CRLF
        COMP    FLAG, 0006
        JUMPZ   PRINT_COUNT
```

```
        RETURN

PRINT_BS
        COMP    POINTER, 00C0
        RETURNC
        LOAD    FLAG, 0000
        RETURN

PRINT_BANNER
        COMP       POINTER, 0098
        RETURNC
        LOAD       FLAG, 0002
        RETURN

PRINT_PROMPT
        COMP       POINTER, 00A4
        RETURNC
        LOAD       FLAG, 0000
        RETURN

PRINT_HOMETOWN
        COMP       POINTER, 00BD
        RETURNC
        LOAD       POINTER, 0098
        LOAD       FLAG, 0002
        RETURN

PRINT_CRLF
        COMP       POINTER, 00C2
        RETURNC
        LOAD       POINTER, 0098
        LOAD       FLAG, 0002  ;prompt
        RETURN

PRINT_COUNT
        COMP    POINTER, 00C4
        RETURNC
        LOAD    POINTER, 00C0
        LOAD    FLAG, 0005
        RETURN

GOT_RXRDY
        COMP       FLAG, 0000
        RETURNNZ
        INPUT      DATA, 0000
        COMP       DATA, 0000 ; check if nothing
        RETURNZ

        COMP       DATA, ASCII_AS
        JUMPZ      HOMETOWN
        COMP       DATA, ASCII_BS
        JUMPZ      BACKSPACE
        COMP       DATA, ASCII_CR
        JUMPZ      NEWLINEFEED
        COMP       DATA, ASCII_AT
        JUMPZ      ATSYMBOL
        ;drop thru, echo char
        ADD        CHAR_COUNT, 0001
        OUTPUT     DATA, 0000
```

```
        COMP       CHAR_COUNT, 0028
        RETURN

HOMETOWN
        LOAD       FLAG, 0003
        LOAD       POINTER, 00A4
        LOAD       TEMP, ASCII_NL
        OUTPUT     TEMP, 0000
        LOAD       CHAR_COUNT, 0000
        RETURN

BACKSPACE
        COMP       CHAR_COUNT, 0000
        RETURNZ
        LOAD       FLAG, 0004
        LOAD       POINTER, 00BD
        LOAD       TEMP, ASCII_NL
        OUTPUT     TEMP, 0000
        SUB        CHAR_COUNT, 0001
        RETURN

NEWLINEFEED
        LOAD       FLAG, 0005
        LOAD       POINTER, 00C0
        LOAD       TEMP, ASCII_NL
        OUTPUT     TEMP, 0000
        LOAD       CHAR_COUNT, 0000
        RETURN

ATSYMBOL
        CALL       BIN2ASCII
        LOAD       FLAG, 0006
        LOAD       POINTER, 00C2
        LOAD       TEMP, ASCII_NL
        OUTPUT     TEMP, 0000
        LOAD       CHAR_COUNT, 0000
        RETURN

; MAIN
main
        OUTPUT  LEDS, 0001      ; walk 1 on LEDs
        RL      LEDS
        CALL    WAIT
        JUMP    main

; interrupt service routine vectored through 0FFE
        ADDRESS 0FFE
        JUMP    ISR
        END
```

```
//*************************************************************//
      // 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 Spring 2018
      // Project name: Project 4 Full UART with TSI
      // File name: Proj4_top.v
      //                                                               //
      // Created by Bryan Linares on 5/8/18
      // Copyright © 2018 Bryan Linares. All rights reserved.
      //                                                               //
      // Abstract: UART Project Top
      // Edit history: 5/10  Revisions 4.0
      //                                                               //
      // 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 sources 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 //
      //*************************************************************//
`timescale 1ns / 1ps

      module Proj4_460_Top(clk, rst, baud, eight, parity_en, ohel, rx, tx, led );

        input    clk, rst;
        input    [3:0] baud;
        input    eight, parity_en, ohel;
        input    rx; //LOC=C4
        output   tx; //LOC=D4
        output   [15:0] led;

        wire     clk_w, rst_w;
        wire     [3:0] baud_w;
        wire     eight_w, parity_en_w, ohel_w;
        wire     rx_w;
        wire     [15:0] led_w;

        UART  core     (.clk(clk_w), .rst(rst_w), .baud(baud_w), .eight(eight_w),
                        .parity_en(parity_en_w), .ohel(ohel_w), .rx(rx_w), .tx(tx_w),
                        .led(led_w));

        TSI   uart_tsi   (.clk_i(clk), .rst_i(rst),  .baud_i(baud), .eight_i(eight),
              .parity_en_i(parity_en), .ohel_i(ohel), .rx_i(rx), .tx_i(tx_w),
              .led_i(led_w),

              .clk_o(clk_w), .rst_o(rst_w),  .baud_o(baud_w), .eight_o(eight_w),
                        .parity_en_o(parity_en_w), .ohel_o(ohel_w), .rx_o(rx_w), .tx_o(tx),
                        .led_o(led));
      endmodule
```

```
//*************************************************************//
      // 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 Spring 2018
      // Project name: Project 4 Full UART with TSI
      // File name: UART.v
      //                                                               //
      // Created by Bryan Linares on 5/8/18
      // Copyright © 2018 Bryan Linares. All rights reserved.
      //                                                               //
      // Abstract: UART Module
      // Edit history: 5/10  Revisions 4.0
      //                                                               //
      // 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 sources 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 //
      //*************************************************************//
      `timescale 1ns / 1ps

      module UART( clk, rst, baud, eight, parity_en, ohel,
                   rx, tx, led);

        input clk, rst;
        input eight, parity_en, ohel;
        input rx;
        input [3:0] baud;
        output tx;
        output reg [15:0] led;

        wire rst_s;
        wire TXRDY, RXRDY, PERR, OVF, FERR;
        wire interrupt, interrupt_ack;
        wire [15:0] port_id;
        wire read_strobe, write_strobe;
        wire [15:0] data;
    wire [15:0] writes, reads;
        wire [7:0] in_port;
    wire [7:0]rxdata;

    ///////////////Baud decoder///////////////count clocks for bit time
        reg [18:0] k;
        always @(*)
          case(baud)
            4'b0000: k = 333333 - 1;   // baud rate = 300 bits per
            4'b0001: k = 83333  - 1;   // baud rate = 1200
            4'b0010: k = 41677  - 1;   // baud rate = 2400
            4'b0011: k = 20833  - 1;   // baud rate = 4800
            4'b0100: k = 10417  - 1;   // baud rate = 9600
            4'b0101: k = 5208   - 1;   // baud rate = 19200
```

```
            4'b0110: k = 2604   - 1;   // baud rate = 38400
            4'b0111: k = 1736   - 1;   // baud rate = 57600
            4'b1000: k = 868    - 1;   // baud rate = 115300
            4'b1001: k = 434    - 1;   // baud rate = 230400
            4'b1010: k = 217    - 1;   // baud rate = 460800
            4'b1011: k = 109    - 1;   // baud rate = 921600
            default: k = 333333 - 1;   // baud rate = 300    switches down
          endcase

        wire RXRDY_pulse, TXRDY_pulse;
        assign uart_int = RXRDY_pulse | TXRDY_pulse;
        assign in_port = (port_id == 16'h0001) ? {3'b000, OVF, FERR, PERR, TXRDY, RXRDY} : rxdata;

        AISO     aiso (.clk(clk), .reset(rst),
            .reset_s(rst_s));
        rsflop   intrflop (.clk(clk), .rst(rst_s), .S(uart_int), .R(interrupt_ack),
              .Q(interrupt));
        UART_tx txengine ( .clk(clk), .rst(rst_s), .load(writes[0]),
                    .data(data[7:0]), .k(k), .eight(eight), .parity_en(parity_en),
                    .ohel(ohel), .TXRDY(TXRDY), .tx(tx) );

        UART_rx  rxengine ( .clk(clk), .rst(rst_s),
                        .clr(reads[0]),
                        .rx(rx), .eight(eight), .parity_en(parity_en),
                        .ohel(ohel), .k(k),
                        .data(rxdata), .RXRDY(RXRDY), .PERR(PERR),
                        .FERR(FERR), .OVF(OVF));

ped          pulse_rx (.clk(clk), .rst(rst_s), .ped_in(RXRDY),
                    .ped_out(RXRDY_pulse));
        ped          pulse_tx (.clk(clk), .rst(rst_s), .ped_in(TXRDY),
                    .ped_out(TXRDY_pulse));

address_decode addr( .port_id(port_id), .write_strobe(write_strobe),
                  .read_strobe(read_strobe), .writes(writes), .reads(reads));

////////////////////////////TramelBlaze Instantiation//////////////////////////////
tramelblaze_top   tb ( .CLK(clk),   .RESET(rst_s),
                        .IN_PORT({8'b0, in_port}), .INTERRUPT(interrupt),
                        .OUT_PORT(data), .PORT_ID(port_id),
                        .READ_STROBE(read_strobe), .WRITE_STROBE(write_strobe),
                        .INTERRUPT_ACK(interrupt_ack) );
////////////////////////////Set LEDS/////////////////////////////////////////////
        always @ (posedge clk, posedge rst_s)
          if (rst_s)
            led <= 16'b0;
          else if (port_id == 16'h0001 && write_strobe)
            led <= data;

      endmodule
```

```
//*****************************************************************//
      // 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 Spring 2018
      // Project name: Project 4 Full UART with TSI
      // File name: aiso.v
      //                                                               //
      // Created by Bryan Linares on 5/8/18
      // Copyright © 2018 Bryan Linares. All rights reserved.
      //                                                               //
      // Abstract: Asynch In Synch out for Reset
      // Edit history: 5/10  Revisions 4.0
      //                                                               //
      // 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 sources 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 //
      //*****************************************************************//
       //          Notes:   The module has two flops that are connected to
      //        the same clock. Module takes an asynchronous
      //                                  signal in and then output a synchronous signal.
      //                                          Synchs release of reset to all flops to avoid metastability.
      ////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

      module AISO(clk, reset, reset_s);

        input    clk, reset;
        output   reset_s;
        reg      inreg, outreg;

        always @ (posedge clk, posedge reset)
          if (reset) {inreg, outreg} <= {1'b0, 1'b0};
          else       {inreg, outreg} <= {1'b1, inreg};

        assign reset_s = ~outreg;
      endmodule
```

```
//*****************************************************************//
      // 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 Spring 2018
      // Project name: Project 4 Full UART with TSI
      // File name: rs_flop.v
      //                                                               //
      // Created by Bryan Linares on 5/8/18
      // Copyright © 2018 Bryan Linares. All rights reserved.
      //                                                               //
      // Abstract: Set Reset Flop
      // Edit history: 5/10  Revisions 4.0
      //                                                               //
      // 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 sources 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 //
      //*****************************************************************//
      `timescale 1ns / 1ps

      module rsflop(clk, rst, S, R, Q);

        input    clk, rst, 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
```

```
//*****************************************************************//
        // 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 Spring 2018
        // Project name: Project 4 Full UART with TSI
        // File name: ped.v
        //                                                               //
        // Created by Bryan Linares on 5/8/18
        // Copyright © 2018 Bryan Linares. All rights reserved.
        //                                                               //
        // Abstract: Pos Edge Detect
        // Edit history: 5/10  Revisions 4.0
        //                                                               //
        // 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 sources 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 //
        //*****************************************************************//

        `timescale 1ns / 1ps

        module ped(clk, rst, ped_in, ped_out);

    input clk, rst;
         input ped_in;
         output ped_out;
         wire ped_out;
         reg delay;

         always@(posedge clk, posedge rst)
         if(rst)
         delay <= 1'b0;
         else
         delay <= ped_in;

         assign ped_out = ~delay & ped_in;

        endmodule
```

```
//*********************************************************************//
        // Class: CECS 460 Spring 2018
        // Project name: Project 4 Full UART with TSI
        // File name: address_decode.v
        //                                                   //
        // Created by Bryan Linares on 5/8/18
        // Copyright © 2018 Bryan Linares. All rights reserved.
        //                                                   //
        // Abstract: Address Decoder , generates read and write enables
        // Edit history: 5/15  Revision 4.0
        //                                                   //
        // 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 sources 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 address_decode(
    input [3:0] port_id,
    input write_strobe,
    input read_strobe,
    output reg [15:0] writes,
    output reg [15:0] reads
    );
      always@(*)begin
      writes = 0;
      reads = 0;
      case({read_strobe,write_strobe,port_id})
        6'b01_0000: begin
                writes = 16'h0001; reads  = 16'h0000;
                end
        6'b01_0001: begin
                writes = 16'h0002; reads  = 16'h0000;
                end
        6'b10_0000: begin
                writes = 16'h0000; reads  = 16'h0001;
                end
        6'b10_0001: begin
                writes = 16'h0000; reads  = 16'h0002;
                end
        default: begin
                 writes = 16'b0; reads  = 16'b0;
               end
        endcase
      end
endmodule
```

```
// 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 Spring 2018
// Project name: Project 4 Full UART with VIC
// File name: UART_tx.v
//                                                                //
// Created by Bryan Linares on 5/8/18
// Copyright © 2018 Bryan Linares. All rights reserved.
//                                                                //
// Abstract: UART transmit engine
// Edit history: 5/10  Revisions 4.0
//                                                                //
// 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 sources 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 //
//****************************************************************//
`timescale 1ns / 1ps

 module UART_tx(clk, rst, load, data, k, eight, parity_en, ohel, TXRDY, tx );

   input    clk, rst, load;
   input    [7:0] data;
   input    eight, parity_en, ohel;
   input    [18:0] k;
   output   tx;
   output   TXRDY;


      //////////rs flops//////
            reg TXRDY;
            always @ (posedge clk, posedge rst)
              if (rst) TXRDY <= 1'b1;
              else if (done) TXRDY <= 1'b1;
              else if (load) TXRDY <= 1'b0;
              else           TXRDY <= TXRDY;

            reg doit;
            always @ (posedge clk, posedge rst)
              if (rst)  doit <= 1'b0;
              else if (loadd1) doit <= 1'b1;
              else if (done)   doit <= 1'b0;
              else             doit <= doit;

            reg loadd1;
            always @ (posedge clk, posedge rst)
              if (rst) loadd1 <= 1'b0;
              else     loadd1 <= load;

            reg [7:0] ldata;
            always @ (posedge clk, posedge rst)
```

```
        if(rst) ldata <= 8'b0;
        else if (load) ldata <= data;

      reg [3:0] bitCountReg;
      reg [3:0] bit_count;
      always @(*)
        case ({doit, btu})
          2'b00: bit_count = 4'b0;
          2'b01: bit_count = 4'b0;
          2'b10: bit_count = bitCountReg;
          2'b11: bit_count = bitCountReg + 1;
          2'b11: bit_count = bitCountReg + 1;
        endcase

      assign done = (bitCountReg == 11);
      always @(posedge clk, posedge rst)
        if (rst) bitCountReg <= 4'b0;
        else     bitCountReg <= bit_count;

      reg [18:0] bitTimeCountReg;
      reg [18:0] btc;
      always @(*)
        case({doit, btu})
          2'b00: btc = 19'b0;
          2'b01: btc = 19'b0;
          2'b10: btc = bitTimeCountReg + 1;
          2'b11: btc = 19'b0;
        endcase

      assign btu = (bitTimeCountReg == k);
      always @(posedge clk, posedge rst)
        if (rst) bitTimeCountReg <= 19'b0;
        else     bitTimeCountReg <= btc;

reg bit9, bit10;
      always @(*)
        case({eight, parity_en, ohel})
          3'b000 : {bit10, bit9} = 2'b11;              // 7N1
          3'b001 : {bit10, bit9} = 2'b11;              // 7N1
          3'b010 : {bit10, bit9} = {1'b1, ^ldata[6:0]};          // 7E1
          3'b011 : {bit10, bit9} = {1'b1, ~(^ldata[6:0])};         // 7O1
          3'b100 : {bit10, bit9} = {1'b1, ldata[7]};     // 8N1
          3'b101 : {bit10, bit9} = {1'b1, ldata[7]};     // 8N1
          3'b110 : {bit10, bit9} = {^ldata[7:0], ldata[7]};      // 8E1
          3'b111 : {bit10, bit9} = {~(^ldata[7:0]), ldata[7]};      // 8O1
          default: {bit10, bit9} = 2'b00; //err
        endcase

//////////////////
      //shift register//
      //////////////////
      reg [10:0] sr;
      always @ (posedge clk, posedge rst)
        if (rst) sr <= 11'hFFFF;            // reset to all 1's
        else if (loadd1)
          sr <= {bit10, bit9, ldata[6:0], 2'b01};
        else if (btu)
```

```
        sr <= {1'b1, sr[10:1]}; //1 in
    assign tx = sr[0];
endmodule
```

```
//*************************************************************//
      // 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 Spring 2018
      // Project name: Project 4 Full UART with TSI
      // File name: UART_rx.v
      //                                                            //
      // Created by Bryan Linares on 5/8/18
      // Copyright © 2018 Bryan Linares. All rights reserved.
      //                                                            //
      // Abstract: UART receive engine
      // Edit history: 5/15  Revisions 4.0
      //                                                            //
      // 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 sources 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 //
      //*************************************************************//
////////////////////////////////////////////
///Receive engine adapted from Pong Chu
////////////////////////////////////////////
      `timescale 1ns / 1ps

      module UART_rx(clk, rst, clr, rx, eight, parity_en, ohel, k, data, RXRDY, PERR, FERR, OVF);

        input    clk, rst;
        input    clr;
        input    rx;
        input    eight, parity_en, ohel;
        input    [18:0] k;
        output reg   RXRDY, PERR, FERR, OVF;
        output   [7:0] data;

        reg [1:0] s, next_s;
        reg start, doit;

        always @ (posedge clk, posedge rst)
          if (rst) s <= 2'b0;
          else     s <= next_s;

        always @ (*)
          case(s)
            2'b00:
              begin
                start = 0;  doit  = 0;
                if (~rx) next_s = 2'b01;
                else next_s = 2'b00;
              end
            2'b01:
              begin
                start = 1; doit  = 1;
```

```
            if (rx)next_s = 2'b00;
            else if (~rx && ~btu) next_s = 2'b01;
            else if (~rx && btu) next_s = 2'b10;
            else next_s = 2'b01;
          end
        2'b10:
          begin
            start = 0; doit  = 1;
            if (done) next_s = 2'b00;
            else next_s = 2'b10;
          end
        default:
          begin
            next_s = 2'b00; start  = 0; doit   = 0;
          end
      endcase

    reg [3:0] bit_countReg;
    reg [3:0] bit_count;
    reg [3:0] count;
    always @(*)
      case ({doit, btu})
        2'b00 : bit_count = 4'b0;
        2'b01 : bit_count = 4'b0;
        2'b10 : bit_count = bit_countReg;
        2'b11 : bit_count = bit_countReg + 1;
    endcase
    always @ (posedge clk, posedge rst)
      if (rst)
        bit_countReg <= 4'b0;
      else
        bit_countReg <= bit_count;

    // bit amount changes depending
    assign done = (bit_countReg == count);
always @ (*)
      case ({eight, parity_en})
        2'b00 : count = 9;      // start+ 7 data + stop bit
        2'b01 : count = 10;      // start+ 7 data + parity bit + stop bit
        2'b10 : count = 10;      // start+ 8 data + stop bit
        2'b11 : count = 11;      // start+ 8 data + parity bit + stop bit
        default: count = 9;
      endcase

    reg [18:0] btc_reg;
    reg [18:0] btc;
    reg [18:0] bitTime;
always @(*)
      case({doit, btu})
        2'b00 : btc = 19'b0;
        2'b01 : btc = 19'b0;
        2'b10 : btc = btc_reg + 1;
        2'b11 : btc = 19'b0;
      endcase

always @(posedge clk, posedge rst)
      if (rst) btc_reg <= 19'b0;
      else btc_reg <= btc;
```

```
        always @ (*)
          if (start) bitTime = k >> 1;
          else       bitTime = k;
        assign btu = (btc_reg == bitTime);

 //shift reg
        reg [9:0] shiftRegOut;
        assign sh = btu & ~start;
        always @ (posedge clk, posedge rst)
          if (rst)
            shiftRegOut <= 10'b0;
          else if (sh)
            shiftRegOut <= {rx, shiftRegOut[9:1]};

// keeps bit placements right side consistent
        reg [9:0] remapped_bits;
        always @ (*)
          case ({eight, parity_en})
            2'b00: remapped_bits =  {2'b11, shiftRegOut[9:2]};
            2'b01: remapped_bits =  {1'b1,  shiftRegOut[9:1]};
            2'b10: remapped_bits =  {1'b1,  shiftRegOut[9:1]};
            2'b11: remapped_bits =  shiftRegOut;
          endcase
        assign data = (eight) ? remapped_bits[7:0] : {1'b0, remapped_bits[6:0]};

        reg p_gen;
        reg p_bit;
        reg p_errorcheck;
        always @ (*)
          begin
            if (eight) p_bit = remapped_bits[8];
            else       p_bit = remapped_bits[7];

            case ({eight, ohel})
              2'b00: p_gen = ^remapped_bits[6:0];    //7E1
              2'b01: p_gen = ~^remapped_bits[6:0];   //7O1
              2'b10: p_gen = ^remapped_bits[7:0];    //8E1
              2'b11: p_gen = ~^remapped_bits[7:0];   //8O1
            endcase
            p_errorcheck = (p_bit ^ p_gen);
          end

        reg stopit;
        always @(*)
          case({eight, parity_en})
            2'b00 : stopit = remapped_bits[7];
            2'b01 : stopit = remapped_bits[8];
            2'b10 : stopit = remapped_bits[8];
            2'b11 : stopit = remapped_bits[9];
          endcase

 ///output status flops
        always @(posedge clk, posedge rst)
          if (rst) RXRDY <= 0;
          else if (done)RXRDY <= 1;
          else if (clr) RXRDY <= 0;
```

```
    else RXRDY <= RXRDY;

  always @(posedge clk, posedge rst)
    if (rst) PERR <= 0;
    else if (p_errorcheck & done & parity_en) PERR <= 1;
    else if (clr)  PERR <= 0;
    else PERR <= PERR;

  always@ (posedge clk, posedge rst)
    if (rst) OVF <= 0;
    else if (RXRDY & done) OVF <= 1;
    else if (clr) OVF <= 0;
    else OVF <= OVF;

  always@ (posedge clk, posedge rst)
    if (rst) FERR <= 0;
    else if (~stopit & done) FERR <= 1;
    else if (clr) FERR <= 0;
    else FERR <= FERR;

endmodule
```

```
//****************************************************************//
	// 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 Spring 2018
	// Project name: Project 4 Full UART with TSI
	// File name: tsi.v
	//							//
	// Created by Bryan Linares on 5/8/18
	// Copyright © 2018 Bryan Linares. All rights reserved.
	//							//
	// Abstract: TSI for UART Module
	// Edit history: 5/10  Revisions 4.0
	//							//
	// 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 sources 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 //
	//****************************************************************//
	`timescale 1ns / 1ps

	module TSI(clk_i, rst_i, baud_i, eight_i, parity_en_i, ohel_i, rx_i, tx_i, led_i,
		clk_o, rst_o, baud_o, eight_o, parity_en_o, ohel_o, rx_o, tx_o, led_o);

	  input clk_i;
	  input rst_i;
	  input [3:0] baud_i;
	  input eight_i, parity_en_i, ohel_i;
	  input rx_i;

	  input tx_i;
	  input [15:0] led_i;

	  output  clk_o;
	  output  rst_o;
	  output  [3:0] baud_o;
	  output  eight_o, parity_en_o, ohel_o;
	  output  rx_o, tx_o;
	  output  [15:0] led_o;

	  BUFG BUFG_inst( .O(clk_o),//1-bitoutput:Clockoutput
			.I(clk_i)//1-bitinput:Clockinput
		       );
	  IBUF#(
	    .IBUF_LOW_PWR("TRUE"),//Lowpower(TRUE)vs.performmance(FALSE)settingforrreferencedI/Ostandards
	    .IOSTANDARD("DEFAULT")//SpecifytheinputI/Ostandard
	    ) rst(
	      .O(rst_o),//Bufferoutput
	      .I(rst_i) //Bufferinput(connectdirectlytotop-levelport)
	    );
	  IBUF#(
	    .IBUF_LOW_PWR("TRUE"),
```

```
    .IOSTANDARD("DEFAULT")
    ) baud[3:0](
      .O(baud_o[3:0]),
      .I(baud_i[3:0])
    );
  IBUF#(
    .IBUF_LOW_PWR("TRUE"),
    .IOSTANDARD("DEFAULT")
    ) eight(
      .O(eight_o),
      .I(eight_i)
    );
  IBUF#(
    .IBUF_LOW_PWR("TRUE"),//Lowpower(TRUE)vs.performrmance(FALSE)settingforrreferencedI/Ostandards
    .IOSTANDARD("DEFAULT")//SpecifytheinputI/Ostandard
    ) pen(
      .O(parity_en_o),//Bufferoutput
      .I(parity_en_i) //Bufferinput(connectdirectlytotop-levelport)
    );
  IBUF#(
    .IBUF_LOW_PWR("TRUE"),
    .IOSTANDARD("DEFAULT")
    ) ohel(
      .O(ohel_o),
      .I(ohel_i)
    );
  IBUF#(
    .IBUF_LOW_PWR("TRUE"),
    .IOSTANDARD("DEFAULT")
    ) rx(
      .O(rx_o),
      .I(rx_i)
    );
  OBUF#(
    .DRIVE(12),//Specifytheoutputdrivestrength
    .IOSTANDARD("DEFAULT"),//SpecifytheoutputI/Ostandard
    .SLEW("SLOW") //Specifytheoutputslewrate
    ) tx(
      .O(tx_o), //Bufferoutput(connectdirectlytotop-levelport)
      .I(tx_i) //Bufferinput
    );
  OBUF#(
    .DRIVE(12),
    .IOSTANDARD("DEFAULT"),
    .SLEW("SLOW")
    ) led[15:0](
      .O(led_o[15:0]),
      .I(led_i[15:0])
    );
endmodule
```

A. Revision History

Rev	Description	Initials	Date
3.0	Transmit Addition, Draft of Document	BDL	03/23/18
4.0	Final Preparation	BDL	5/15/1