



# 460 FALL 2019

UART Engine

Chip Specification Document

Zachery Takkesh

015656509

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## CONTENTS

1. INTRODUCTION.....	4
2. REQUIREMENTS .....	4
2.1 Software Design.....	4
2.2 Demonstration.....	4
3. TOP LEVEL DESIGN.....	5
3.1 Description .....	5
3.2 Block Diagram.....	5
3.2.1 Top Level Digram.....	5
3.2.2 Detailed Top Level Diagram.....	6
3.3 Data Flow Description.....	7
3.4 I/O.....	7
3.4.1 Signal Names.....	7
3.4.2 Pin Assignments .....	9
3.5 Software Design.....	9
3.5.1 Description .....	9
3.5.2 Detailed Software Flowchart.....	11
4. TRAMELBLAZE.....	15
4.1 Top Level Block Diagram .....	15
4.2 Detailed Level Block Diagram.....	15
5. UART .....	16
5.1 Description .....	16
5.2 Block Level Diagram.....	16
5.3 Data Flow Description.....	16
5.4 I/O.....	17
6. TRANSMIT ENGINE.....	18
6.1 Description .....	18
6.1.1 Shift Register.....	18
6.1.2 Bit Time Counter .....	18
6.1.3 Bit Counter .....	19
6.1.3 Sr & Do it & Load Flops.....	19
6.2 Block Diagram.....	20

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

6.3 Verification.....	20
6.3.1 Transmit Engine Verification Waveform .....	21
7. RECEIVE ENGINE.....	23
7.1 Description .....	23
7.2 Receive Engine control .....	23
7.2.1 Block Level Diagram .....	23
7.2.2 Bit Time Counter .....	23
7.2.3 Bit Counter .....	24
7.2.4 Finite State Machine.....	24
7.3 Receive Engine Data Path.....	24
7.3.1 Block Level Diagram .....	25
7.3.2 Shift Register.....	25
7.3.3 Remap Function .....	25
7.3.4 RxRdy.....	26
7.3.5 Parity Error.....	26
7.3.6 Framing Error .....	26
7.3.7 Overflow Error.....	26
7.4 Verification.....	26
7.4.1 Rx Engine Waveform Verification.....	27
8. TECHNOLOGY SPECIFIC INSTANTIATION .....	28
8.1 Description .....	28
8.2 Block Diagram.....	28
9. SOURCE CODE.....	28

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## I. INTRODUCTION

This document provides the requirements and technical information for a functional Universal Asynchronous Receiver Transmitter (UART) created using Verilog in Xilinx ISE. This document is for reference of the design and development done during the System-on-Chip Design course led by John Tramel. The design is composed of a Transmit Engine(Tx) and Receive Engine(Rx) The Tx engine architecture is designed with multiple sequential logic blocks. The main block of the Tx engine consists of a shift register that shifts a data package out of the engine one bit at a time. The Rx engine is designed using numerous combinational logic blocks and a Finite State Machine(FSM) that processes serial data coming through the Rx signal. The Baud rate determines the speed of the data being exchanged<sup>1</sup> between the terminal program and device.

Chip Specification will be updated and modified as other components of the UART are fully designed and implemented.

## 2. REQUIREMENTS

The requirements for Project 3 are split into two categories:

Software Design & Demonstration of a Full UART

### 2.1 SOFTWARE DESIGN

The Software for the UART is designed and implemented using Assembly code. The Software design specifications for Project 3 entails:

- i. Display a banner message upon assertion of **reset**
- ii. Display a prompt line for the user to input a character or key.
- iii. On any key press, the key will be displayed on the prompt line.
- iv. Only four keys will trigger a response from the UART block
  - I. “\*”- An input of an Asterisk will display the hometown of the project designer, subsequently a new prompt line is displayed.
  - II. “@”- An input of the At character will display the number of characters the terminal has received from the user prior to the arrival of the “@” key.
  - III. “Enter”- An input of the return key will display the prompt on a new line
  - IV. “BS”- An input of the backspace key will delete the previous inputted character from the Terminal.

### 2.2 DEMONSTRATION

Generated design bit file is loaded to the Nexys4 DDR FPGA board, with the only observable feature being a pattern of walking LEDs. Button Up (BTNU M18) controls the reset of the running program. The Software output is displayed via serial terminal program.

---

<sup>1</sup> Exchange refers to Transmit and Receive of Data

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

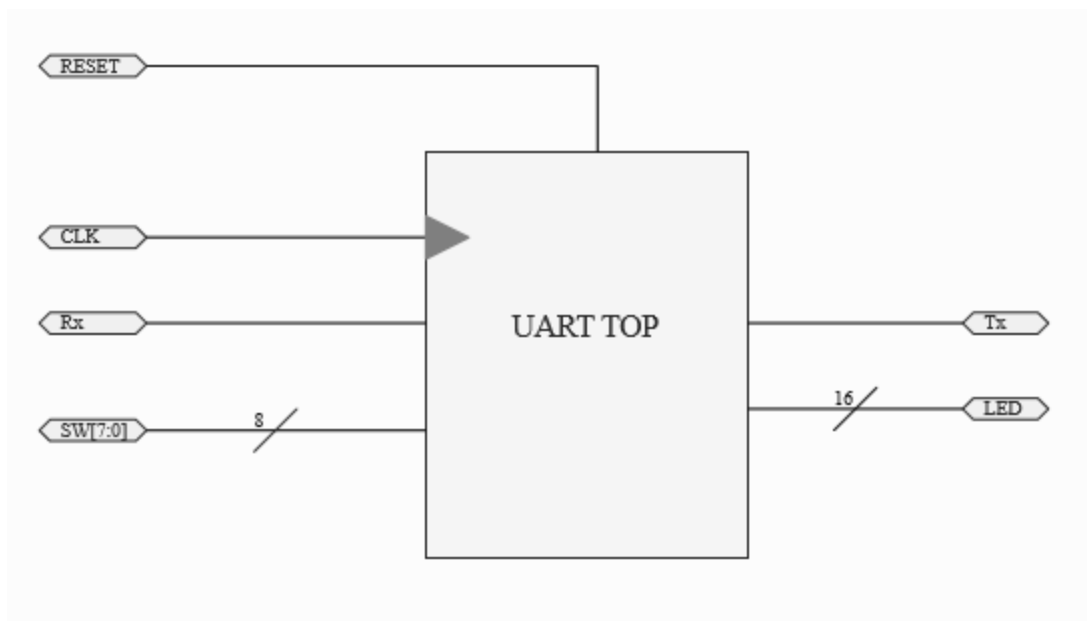
### 3. TOP LEVEL DESIGN

#### 3.1 DESCRIPTION

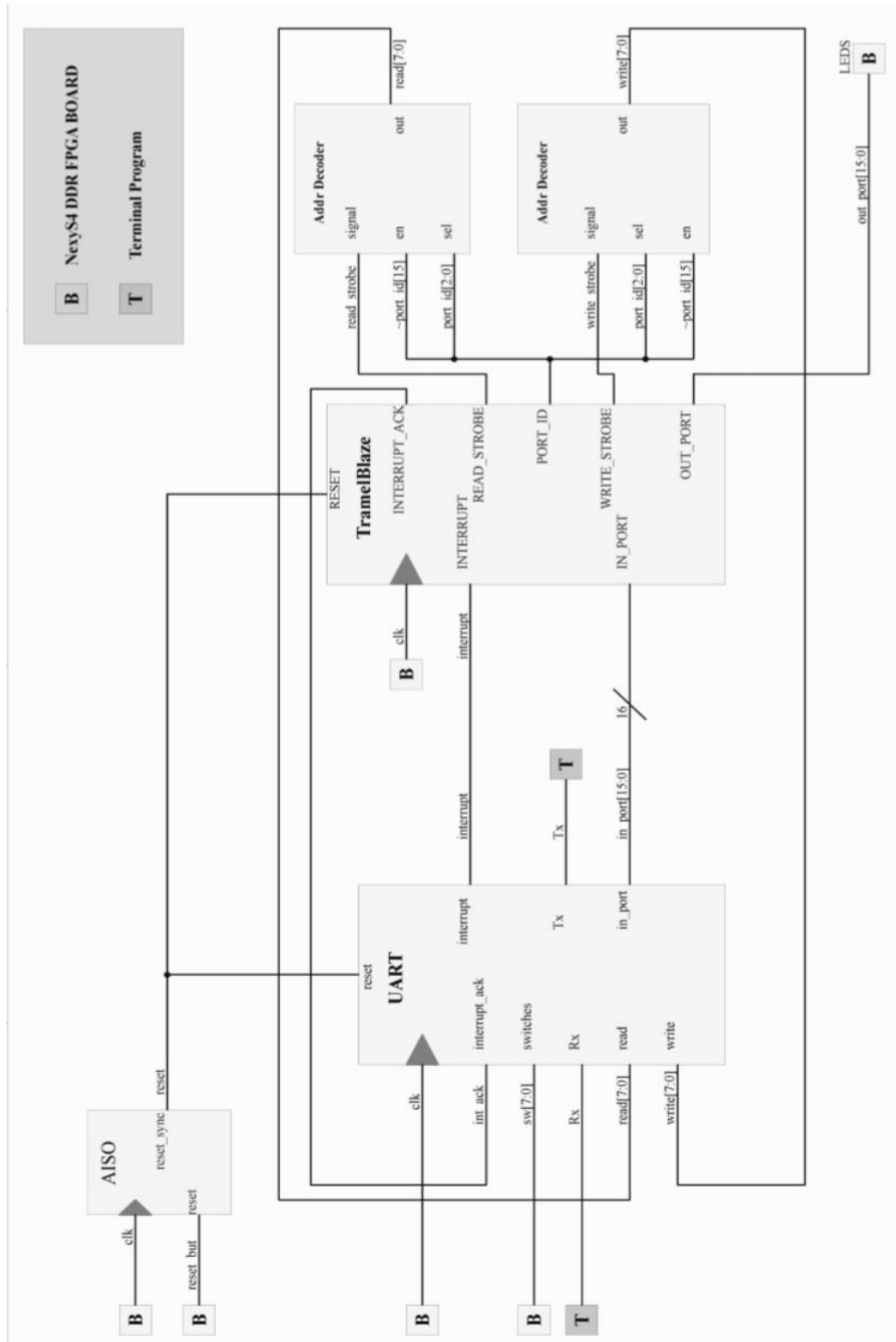
The UART engine is designed with the purpose of Serial communication with a computer. To observe the data being exchanged between the UART and computing device, the serial terminal program, Realterm is used. Serial terminal programs have settings that need to be carefully configured to enable clean and safe data exchanges. These settings include: Baud rate, parity mode enable, even or odd parity, and data size. To ensure that RealTerm will display the correct message, the UART design board configuration must match the settings in RealTerm using the onboard switches. A common error when exchanging data is caused by a mismatch of settings, resulting in no response or unreadable characters between the terminal and FPGA Device.

#### 3.2 BLOCK DIAGRAM

##### 3.2.1 TOP LEVEL DIGRAM



### 3.2.2 DETAILED TOP LEVEL DIAGRAM



Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

### 3.3 DATA FLOW DESCRIPTION

The main Reset signal is delivered to the AISO module upon release of the designated reset button on the NexyS4 DDR to synchronize reset to all modules to prevent metastability within the design. Every other module's reset port takes in the synchronous reset signal from the AISO module. The UART engine contains both Tx and Rx engines for the purpose of using the TramelBlaze processor to execute the UART engine instructions. The TramelBlaze being an externally developed block, is used for driving the LED's in a walking pattern on the NexyS4 DDR board as well. The Address decoder module is used to decode the read and write strobe data to write to specific address locations in memory. The UART configuration signals: BAUD, EIGHT, PEN, and OHEL are controlled by onboard switches and the sixteen LEDs serve as an output.

### 3.4 I/O

#### 3.4.1 SIGNAL NAMES

##### **INPUTS:**

##### I) BAUD

- i) Controls the rate of which data is being transmitted and outputted.
- ii) 4-bit wide signal is decoded to 12 different communication speed rates.

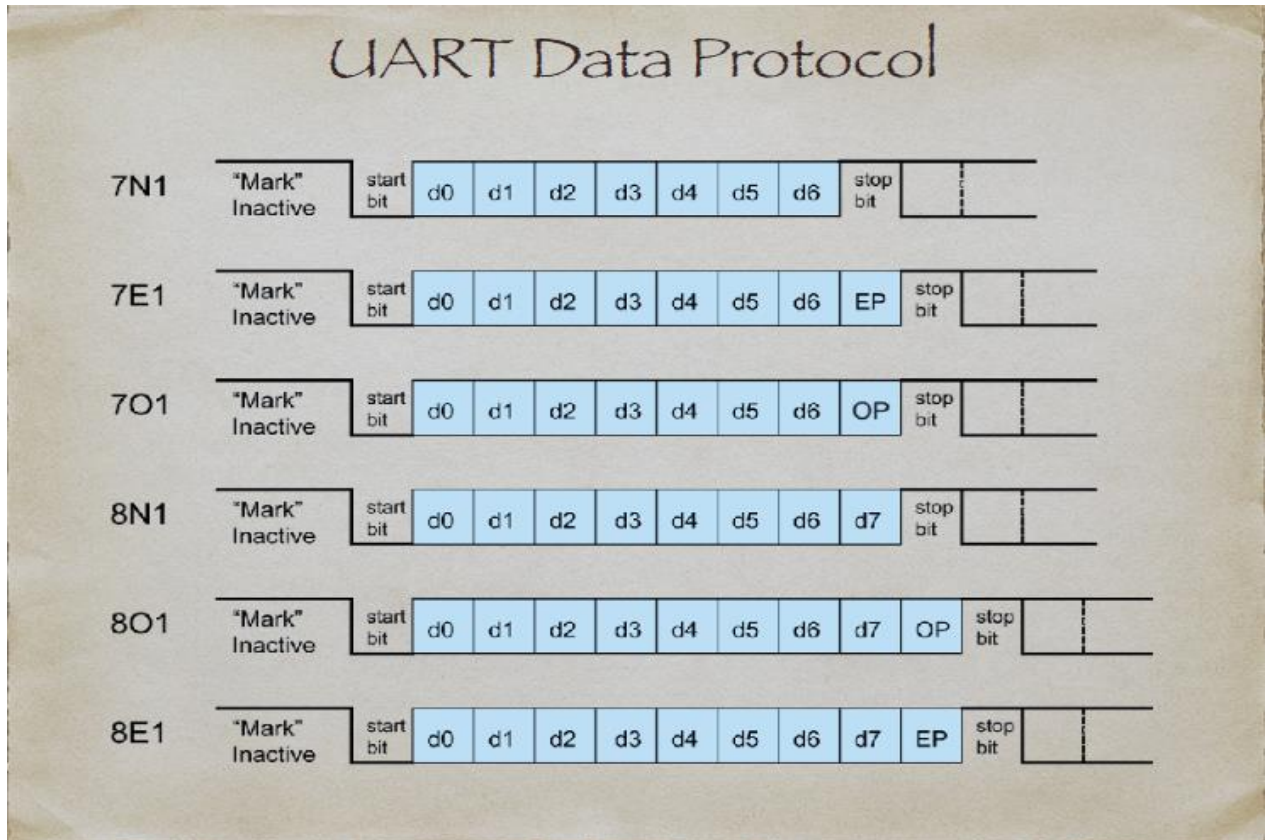
BAUD CASE	DECIMAL COUNT	RATE
0000	333,333	300
0001	83,333	1,200
0010	41,667	2,400
0011	20,833	4,800
0100	10,417	9,600
0101	5,208	19,200
0110	2,604	38,400
0111	1,736	57,600
1000	868	115,200
1001	434	230,400
1010	217	460,800
1011	109	921,600

##### 2) EIGHT

- a) Marks if the data is seven or eight bits.
  - i) If signal is HIGH, data is 8-bits wide.
  - ii) If signal is LOW, data is 7-bits wide.

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

- 3) PEN
  - i) Parity Enable Mode
- 4) OHEL
  - i) If signal is HIGH, odd parity mode is enabled.
  - ii) If signal is LOW, even parity mode is enabled.



- 5) CLK
  - i) Default 100 MHz clock speed from NexyS4 DDR board.
- 6) RESET
  - i) Reset from button up release sent into AISO for synchronous reset for the design.
- 7) Rx
  - i) Signal to receive input data from serial terminal program

### **OUTPUTS:**

- 1) LEDs
  - i) Display walking pattern LEDs.
  - ii) Driven by TramelBlaze out port.
- 2) Tx
  - i) Signal to transmit data to the serial terminal program.



Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

### 3.4.2 PIN ASSIGNMENTS

Signal name	Pin Locations
clk	E3
reset	M18
eight	R15
pen	M13
ohel	L16
baud [0]	R17
baud [1]	T18
baud [2]	U18
baud [3]	R13
Rx	C4
Tx	D4
LED [0]	H17
LED [1]	K15
LED [2]	J13

Signal Name	Pin Locations
LED [3]	N14
LED [4]	R18
LED [5]	V17
LED [6]	U17
LED [7]	U16
LED [8]	V16
LED [9]	T15
LED [10]	U14
LED [11]	T16
LED [12]	V15
LED [13]	V14
LED [14]	V12
LED [15]	V11

## 3.5 SOFTWARE DESIGN

Software design for this Project consists of developing Assemble code to display the NexyS4 DDR onboard LEDs in a walking pattern is addition to outputting the required message of

```

-----
-    Zach Takkesh
-    CECS 460
-    FULL UART
-----

```

Followed by a user input prompt of “Enter a key”. The inputted key will trigger a pre-determined mode, outputting a message to the terminal.

### 3.5.1 DESCRIPTION

Assembly code is composed of five critical sections: Setup, Main loop, Service routines, Interrupt service routines, and Interrupt service routine vector.

The Setup section is responsible for loading ASCII values of the required letters and characters. In this design there are ten variables to reference the start and end addresses for the welcome banner, prompt,

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

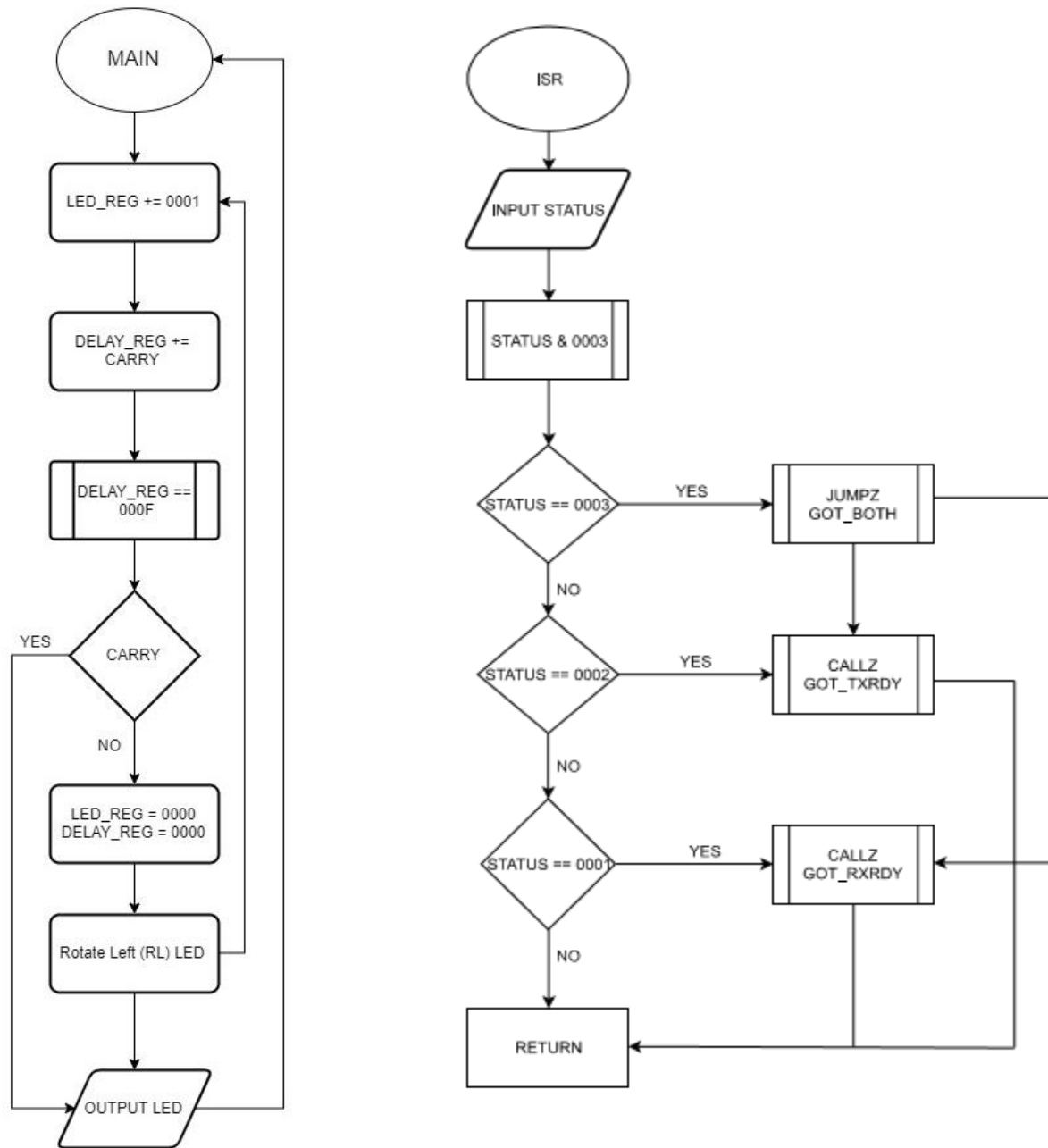
hometown, newline, and backspace. Ten additional registers are utilized for the purpose of subroutines and interrupt service routines processing. Within the initialization block, all previously declared registers are loaded with zero except for the CASE(RC) and LED(R6) register, getting a value of one. There are five subroutines called for the purpose of loading the ASCII values to a defined memory location to be later called to output content stored in the specified memory location. The five subroutines called are: banner\_init, prompt\_init, hometown\_init, crlf\_init, and bs\_init.

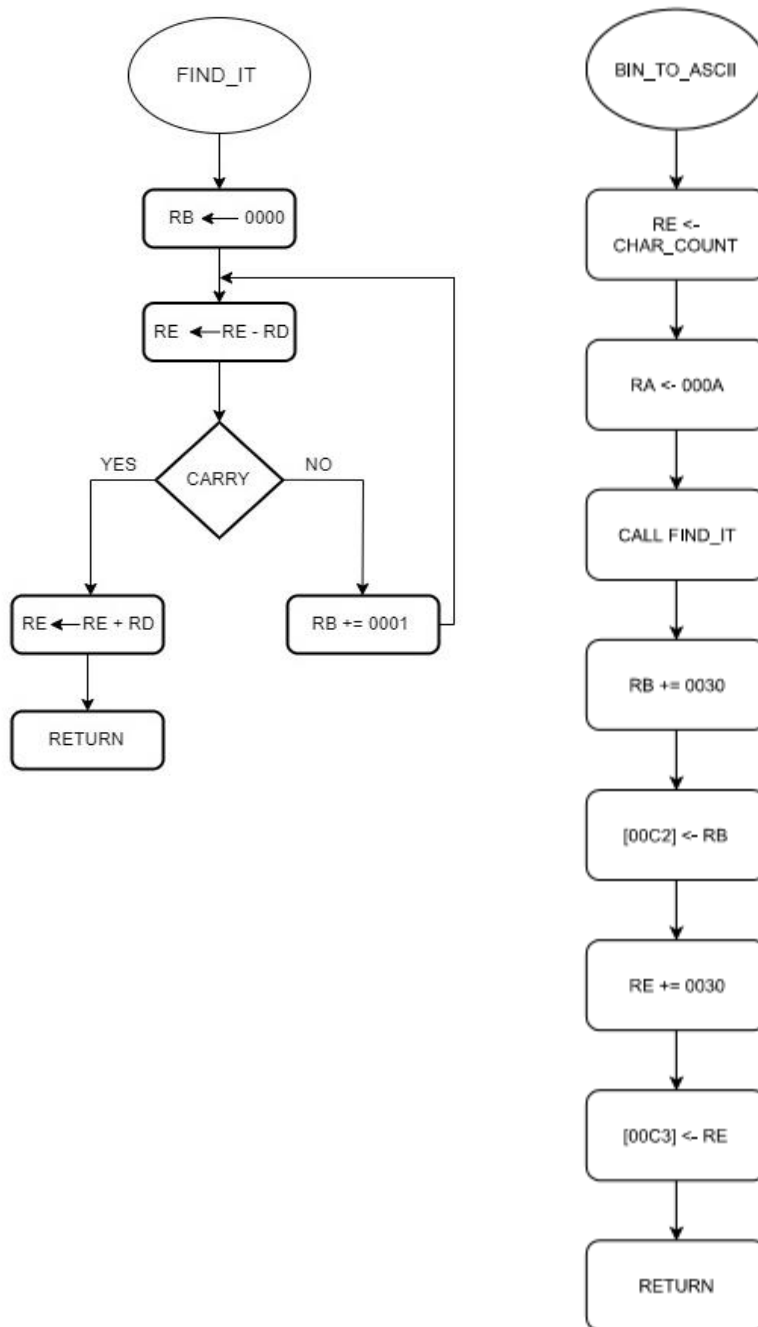
Within the main loop, the algorithm is designed to display a walking pattern through the sixteen available onboard LEDs. The displaying of LEDs symbolizes the software is running within the main loop. To implement the walking LEDs, the rotate left instruction is implemented with the LEDs being sent to memory location 0002. To visually see the LEDs walking with a naked eye, a delay function is needed. A delay register is utilized create a fixed delay between each bit rotation of the LEDs. The main function will continuously loop until a subroutine or Interrupt Service Routine (ISR) is triggered.

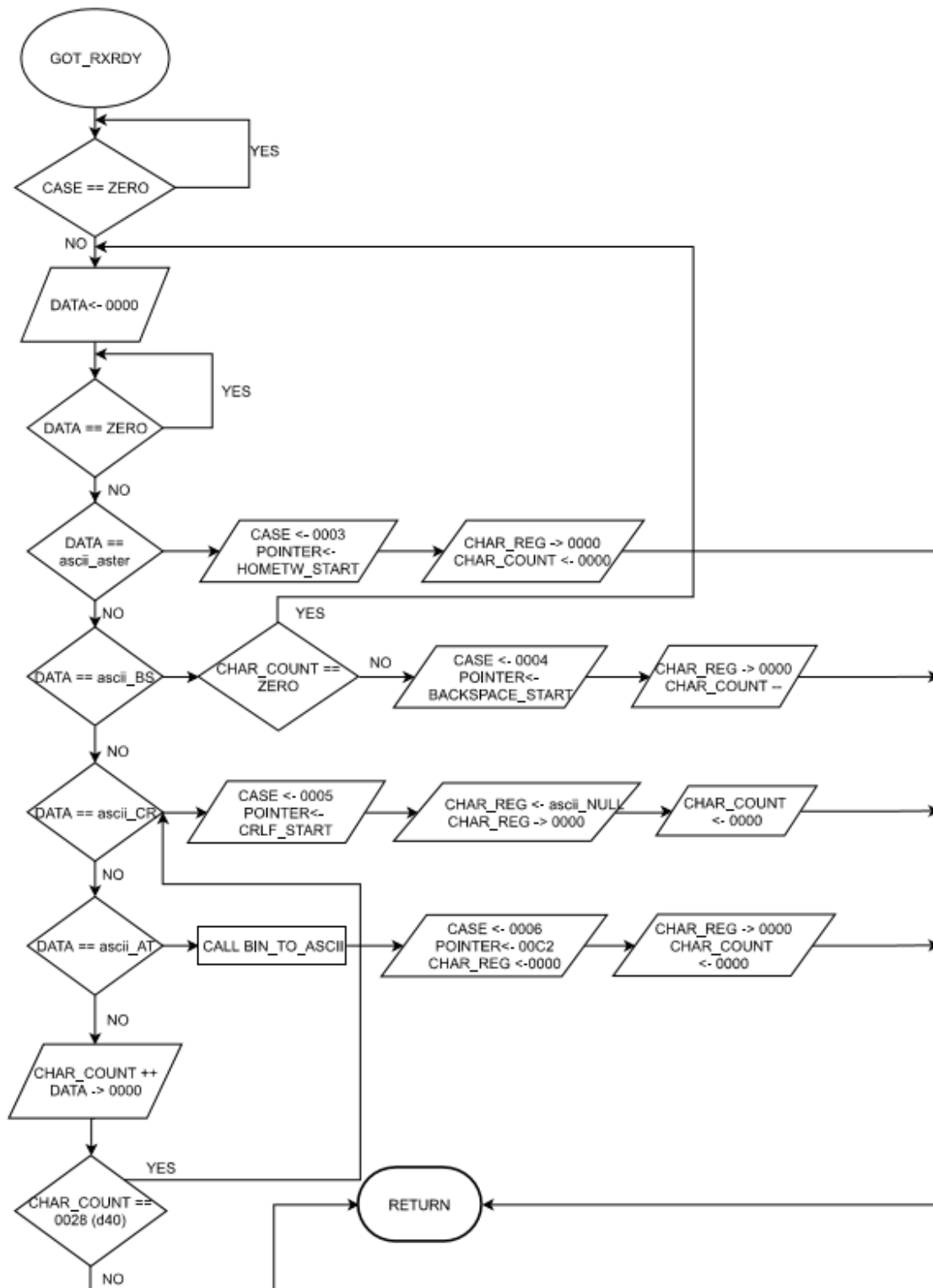
Four Service routines are created to ensure functionality of the design: GOT\_TXRDY, GOT\_RXRDY, BIN\_TO\_ASCII, and FIND\_IT. Serial terminal programs such as PuTTY and RealTerm only reads signals in ASCII format, a function or routine to convert the value on the line count from Binary to ASCII for a correct display of outputs is needed. The BIN\_TO\_ASCII routine converts binary value within the UART engine to an ASCII value by adding 30 to the incoming value and storing ASCII value to a memory address. The FIND\_IT subroutine works with the BIN\_TO\_ASCII function to convert each value to the correct decimal position. The GOT\_TXRDY routine extracts data from a pointer memory location based on the current case value evaluated by the service routine. The GOT\_RXRDY routine will evaluate the result of the GOT\_TXRDY routine and user input and select a case to be displayed on the terminal program. Case 1 will print out the initial banner, Case 2 will prompt the user for an input, Case 3 will print the designer's hometown, Case 4 references the backspace key and deletes the user's input, Case 5 will display a new line, and Case 6 will respond with the number of inputted characters from the user. The logic flow will execute cases 1, 3, 4, 5, and 6 before executing a new prompt with case 2. All 4 routines need to work seamlessly to correctly convert and output data.

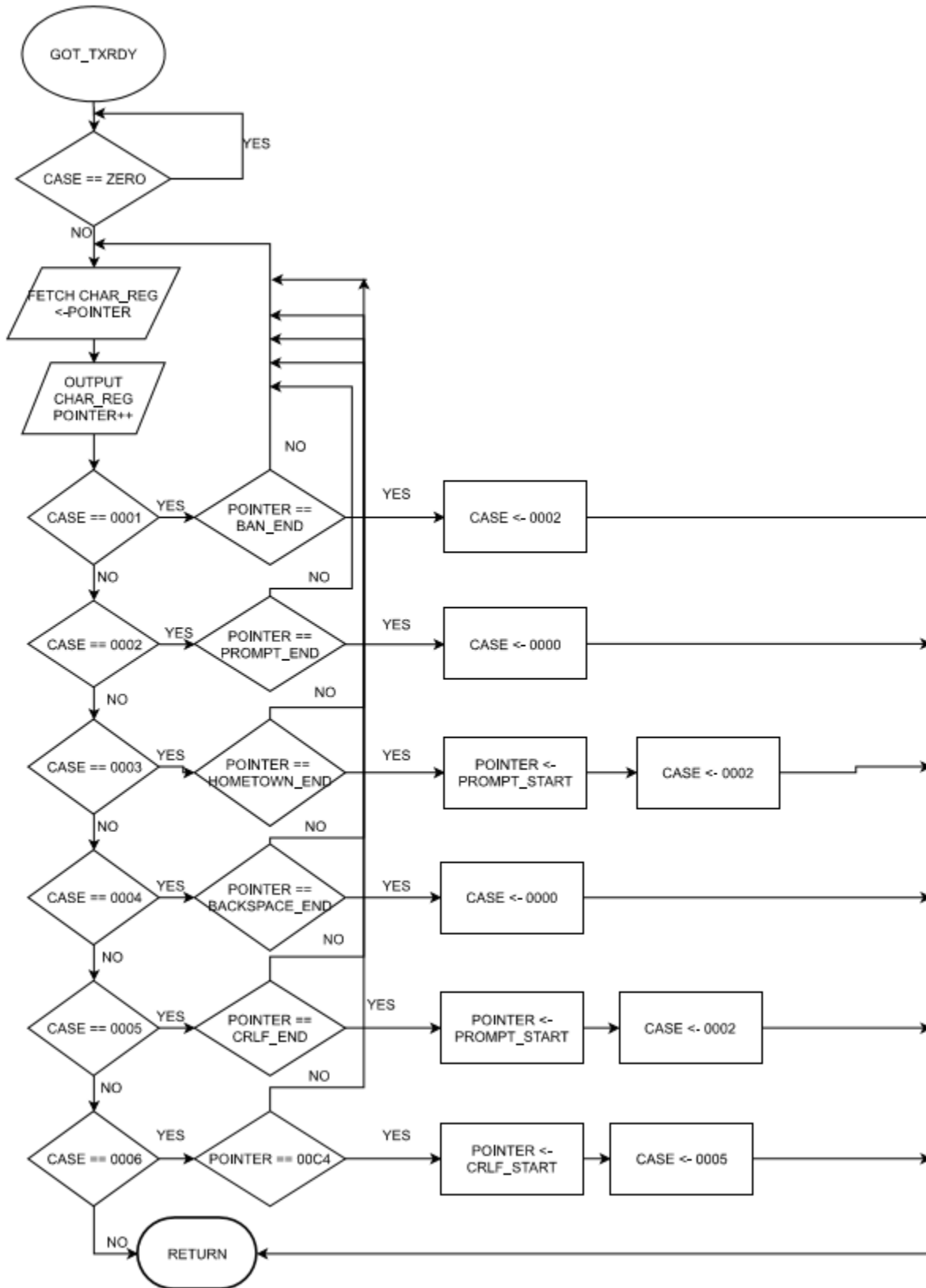
Interrupt Service Routine (ISR) and Vector(ISRV) are another critical section of the Assembly code. The ISR in our design is used for sampling the incoming TxRdy and RxRdy signals. The sampled signal is placed at memory address 0001 for the status register. When the status register(R1) has value of 0001 and 0002, GOT\_RXRDY and GOT\_TXRDY routines are called respectively. When status register has a value of 0003, indicating both Tx and Rx signals are active, both routines are called. The routines will return an interrupt enable value. The ISR vector is contained at instruction memory location 0FFE documented on line 134 in the TramelBlaze package.

### 3.5.2 DETAILED SOFTWARE FLOWCHART





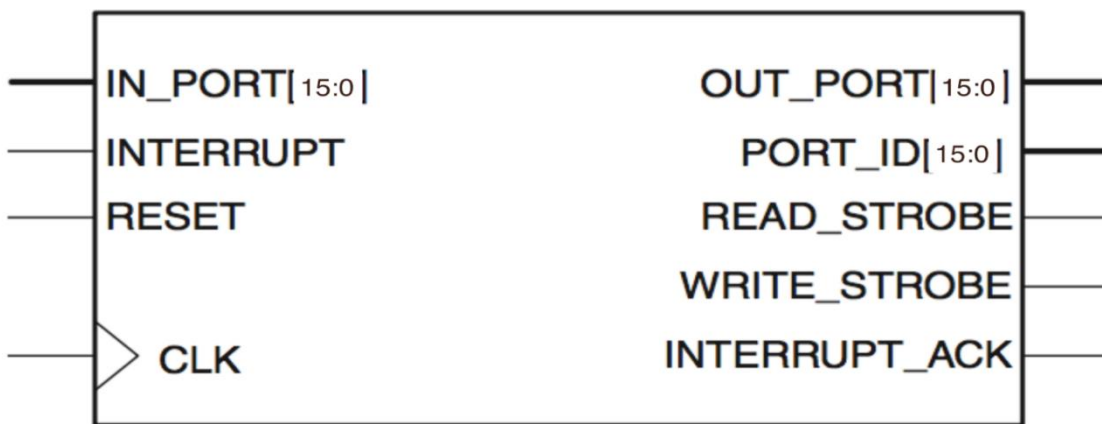




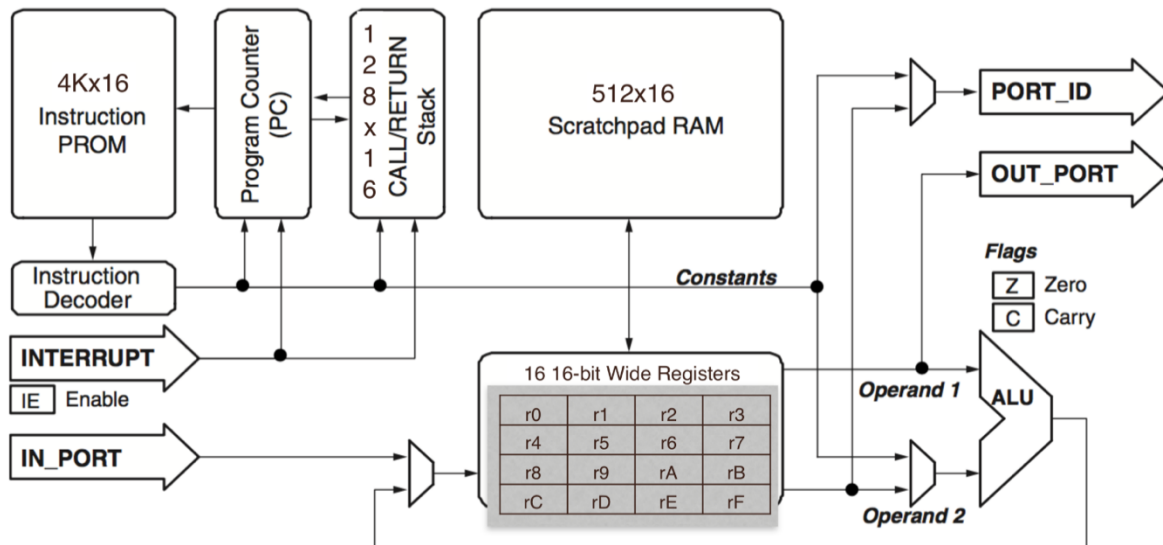
## 4. TRAMELBLAZE

The TramelBlaze is a 16-bit emulator of the 8-bit PicoBlaze processor. The TramelBlaze software is an external developed block integrated into this project with the purpose of serving as a processor for the UART engine. There are three memory blocks instantiated within the TramelBlaze: 4096x16 Instruction ROM, 128x19 CALL/RETURN Stack RAM, and 512x16 Scratchpad RAM. The ROM is used to store the instructions written in assembly and generated from a Python assembler. There are two ways to instantiate the sim file: through COE or reading through directory. The 128x16 stack RAM is used to track return addresses for subroutines and interrupt handler. The 512x16 scratchpad RAM is used for immediate value. There are 16 16-bit wide built-in registers used for any ALU operations. The assembler for the software code is written Python and it generates necessary memory COE core files required for building a Xilinx code ROM.

### 4.1 TOP LEVEL BLOCK DIAGRAM



### 4.2 DETAILED LEVEL BLOCK DIAGRAM



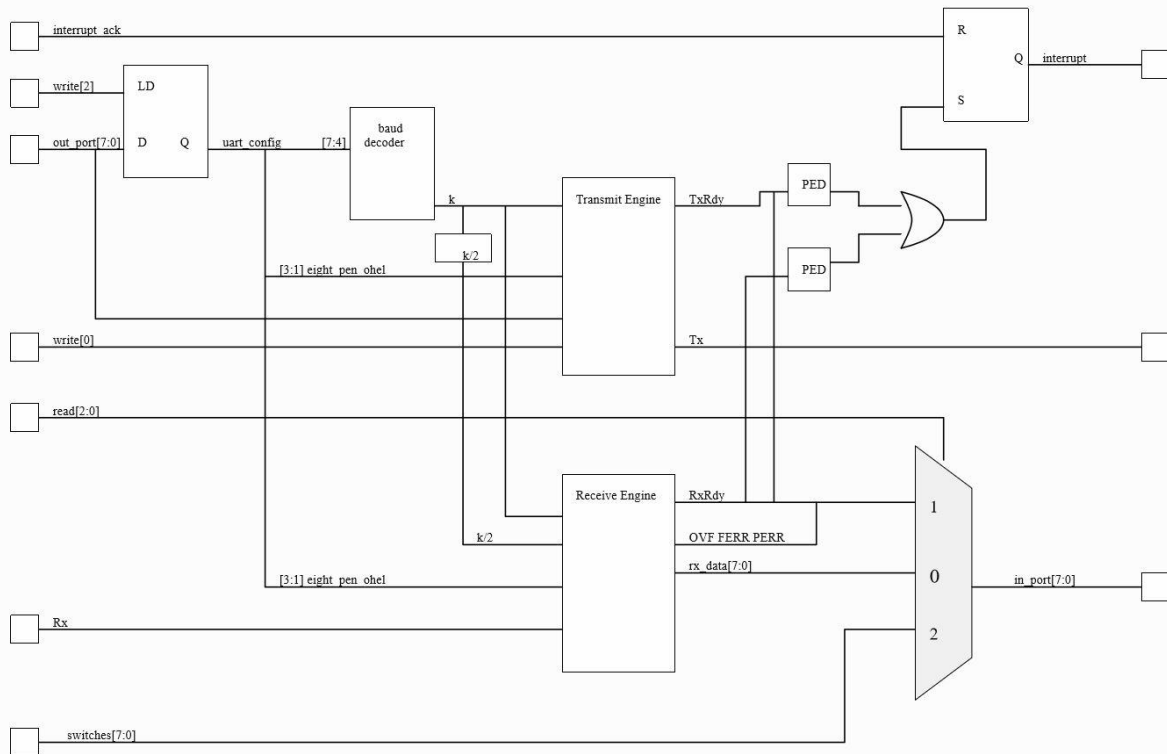
Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## 5. UART

### 5.1 DESCRIPTION

The UART serves as a wrapper to connect the Transmit and Receive engines together for terminal communication. The baud decoder module is instantiated inside the design to synchronize the speed of data exchanging between the engines. The UART is designed to combine the logical blocks needed for communication between the TramelBlaze Processor and serial terminal program.

### 5.2 BLOCK LEVEL DIAGRAM



### 5.3 DATA FLOW DESCRIPTION

The UART protocol configuration bits are evaluated during initialization and stored to a register. The register provides the destination for all configurations within the UART engine. The Baud rate decoder creates two speed signals,  $k/2$  and  $k$  sent to the Tx and Rx engine. Multiplexor is utilized to select data between the on-board switches along with the UART data and status signals. A positive-edge detect circuit is used to generate interrupt signal on the TxRdy and RxRdy lines.



Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## 5.4 I/O

### **INPUTS:**

- 1) **interrupt ack**
  - a. Signal from the TramelBlaze to generate an interrupt via RS flop.
- 2) **write[2]**
  - a. Scalar signal from the address decoder to load the UART configurations.
- 3) **write[0]**
  - a. Scalar signal from the address decoder for the Tx engine load indicator.
- 4) **Rx**
  - a. Signal to receive inputted serial data for the Rx engine.
- 5) **Switches**
  - a. Eight-bit signal corresponding to the onboard switches for UART configuration

switch[7]	switch[6]	switch[5]	switch[4]	switch[3]	switch[2]	switch[1]	switch[0]
baud[3]	baud[2]	baud[1]	baud[0]	eight	pen	ohel	empty/null

### **OUTPUTS:**

- 1) **Interrupt**
  - a. Signal generated through the RS flop in the external block of TramelBlaze.
- 2) **Tx**
  - a. Signal to transmit serial data to terminal program.
- 3) **In port**
  - a. Eight-bit signal that gets concatenated with an 8'b0 for the MSB to accommodate the TramelBlaze's 16-bit in port channel.

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## 6. TRANSMIT ENGINE

### 6.1 DESCRIPTION

The Transmit Engine receives parallel data with the TrameBlaze and Receive (Rx) Engine. Tx Engine loads data to the shift register and outputs the least significant bit (LSB) to the Tx signal. The size of the data from the Tx Engine ranges from 9 to 11 bits with the values determined by the loaded data and signal. The first (start) bit signal loaded into the Tx Engine is always LOW while the last (end) bit is always HIGH. The start and end bit configuration ensures the receive port can consistently recognize the stream of data.

#### 6.1.1 SHIFT REGISTER

A Shift Register with a Parity Decoder module is essential to the functionality of the Tx Engine. The register shifts data out with a dependency on the Parity Decode signal. When reset is asserted the value sets to 11'b111\_1111\_1111. The reset/default value is set to HIGH to notify the receiver that the Engine is ready to transmit data. The transmitted data is acknowledged by the TrameBlaze which will convert the data into Hexadecimal values for the assembler which then converts the value to the ASCII encoding standard.

#### Parity Decoder Chart:

eight	pen	ohel	D [10]	D [9]
0	0	0	1	1
0	0	1	1	1
0	1	0	1	EP
0	1	1	1	OP
1	0	0	1	D [7]
1	0	1	1	D [7]
1	1	0	EP	D [7]
1	1	1	OP	D [7]

EP – Even Parity

OP – Odd Parity

To analyze the loaded data parity bit correctly a bit-wise exclusive-or (XOR) gate is implemented. If the output of the XOR gate is 1, an even parity is generated. If the output of the XOR gate is 0, an odd parity is generated.

#### 6.1.2 BIT TIME COUNTER

Bit-time Counter informs the Tx Engine the shifting rate of the current loaded data. The bit time counter is designed using a multiplexor register that increments a comparator variable.

The comparator evaluates the counted value to the selected baud rate controlling the speed of the data transmission. The signal (BTU) is HIGH active and notifies the shift register to shift out the current data when BTU signal is '1'.

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

### 6.1.3 BIT COUNTER

Bit Counter is designed in a similar manner of the Bit Time Counter function. The Bit counter computes a sum of all the bits being transmitted, the data shifted is twelve-bit wide because the comparator variable counts between eleven to zero. After twelve bits finish data transmission, a flag signal is created to inform of the data transmission. A delay between Bit Counter and TxRdy flop is needed to ensure the data is accurate. The signal done\_dI is created to delay the output of the Bit Counter by one clock period.

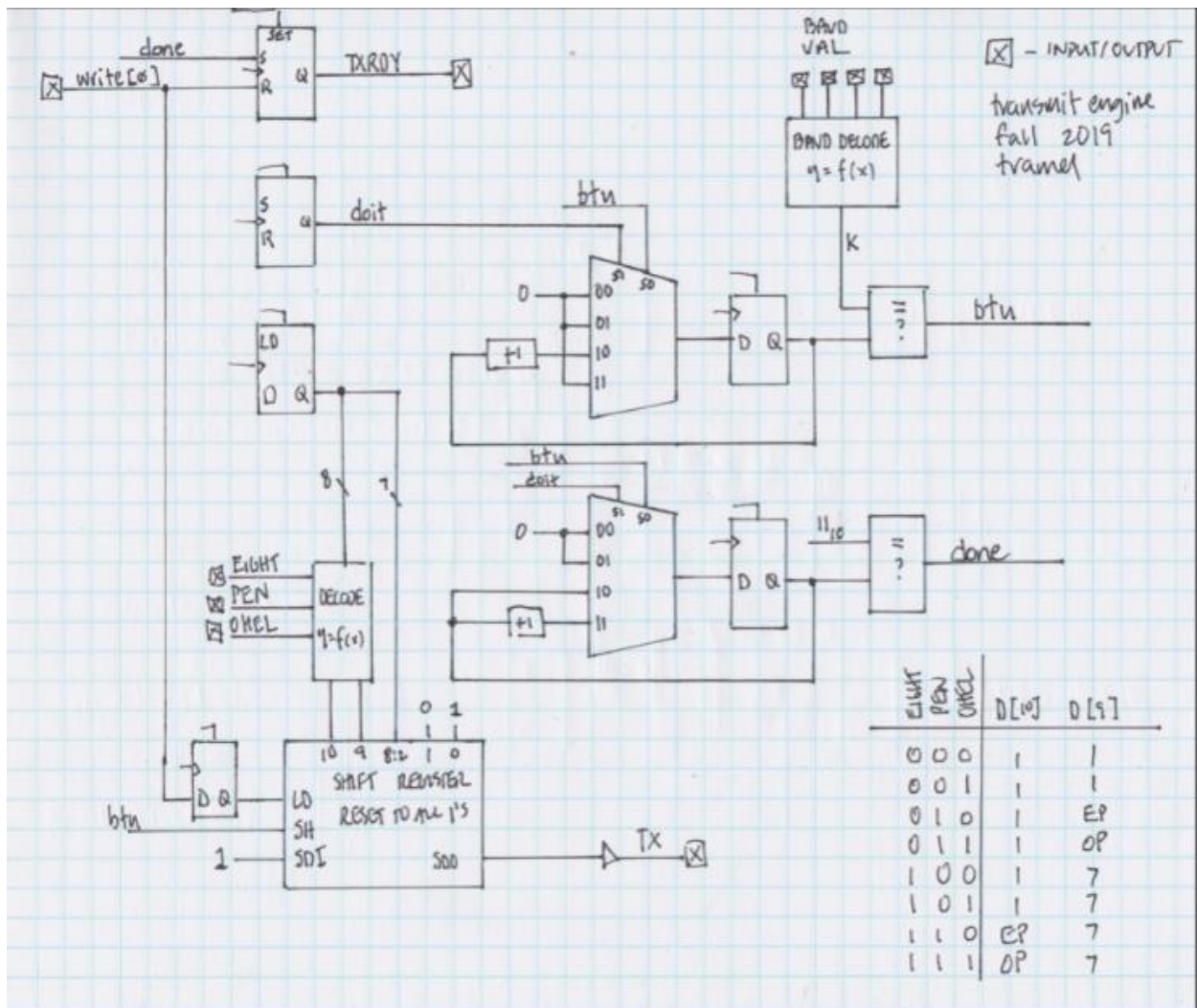
### 6.1.3 SR & DO IT & LOAD FLOPS

The SR, DO IT, and LOAD flops are essential to the functionality of the Tx Engine. The SR flop is designed using Sequential logic and a Multiplexor with the purpose of establishing a signal to notify that the Tx Engine is no longer waiting for a process to finish. The TxRdy signal is set and passed to the TxRdy wire when both the done\_dI and load signals are LOW. When the done\_dI signal is HIGH and the load signal is LOW, the TxRdy receives a value of '1'. The other combinations of done\_dI and load set TxRdy with the value of '0' or '?', latter being an either-or value.

The DO IT flop is designed using the same structure as the SR flop but with the purpose of creating a data signal to deliver to the Shift Register. The do\_it signal is passed to itself on a one-bit wire when both the load and done signals are LOW. The do\_it wire receives a '1' when the load signal is HIGH and done signal is LOW. The other combinations of load and done set the do\_it wire with the value of '0' or '?', latter being an either-or value.

The LOAD flop is designed using Sequential logic. The LOAD flop takes the data an external module input register and loads the data of the register to an internal register called load\_data. When reset is asserted load\_data gets cleared with eight bits of 0. When the input load signal is HIGH, the LOAD flop passes data from the out\_port to the load\_data register to later pass to the Shift Register and Parity Decoder. If the input load signal is LOW, the load\_data register receives itself.

## 6.2 BLOCK DIAGRAM



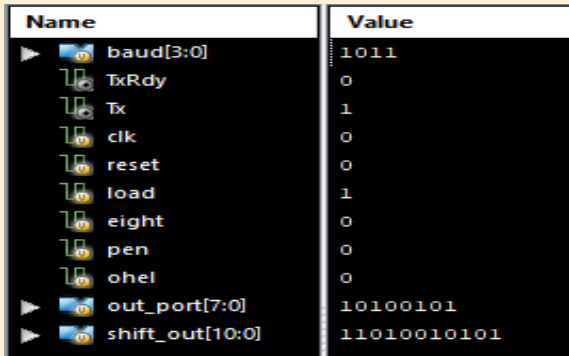
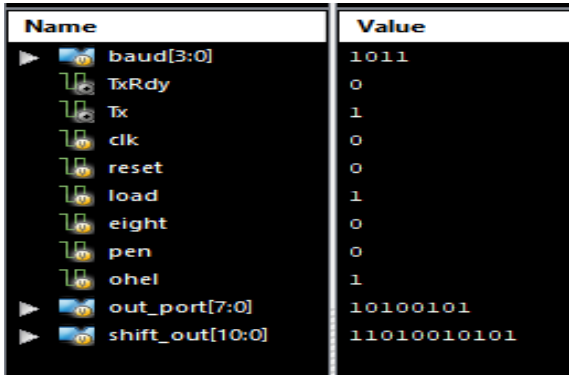
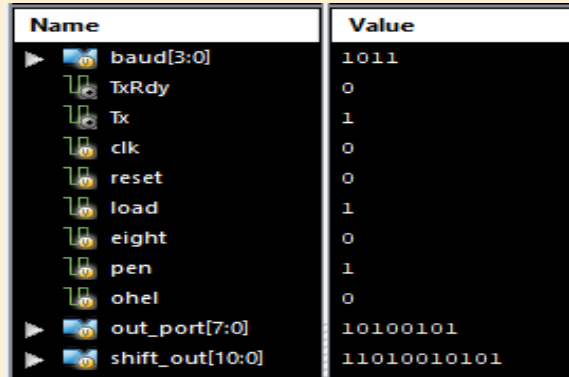
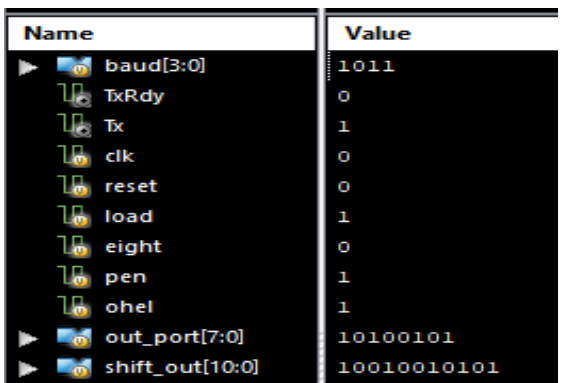
## 6.3 VERIFICATION

Verification is necessary to prove accountability of the many components within the design of the Tx engine. For document version 1.0, the verification operation focuses on both the load and shift registers in the Tx Engine.

To verify the precision of the decode, load and shift registers, the out\_port signal is assigned to the value of 8'b1010\_1010 or 8'hAA. Reset is asserted during the initialization to allow all variables to go to a known state and deasserted after a fixed time to begin the verification process. The Baud rate is set to its maximum speed in the design for easier verification of the Tx Engine. All specific cases for the combination of signals: eight, pen, and ohel are configured to observe and verify the accuracy of the decoded values.

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

### 6.3.1 TRANSMIT ENGINE VERIFICATION WAVEFORM

{eight, pen, ohel}	Waveforms	Description
000		<p>When all signals low, D[10] and [9] are 1's. With the eight signal being low, the size of data is 7 bits configured within bits 2-8.</p>
001		<p>Although the ohel signal is high, the Parity enable(PEN) is low disabling parity mode. The other data remains consistent.</p>
010		<p>Parity mode is active while ohel and eight are low. Informs the parity is even. With eight set low, 7 bits are transmitted as 010_0010. D[9] is 1 to even the parity bits. The other data remains consistent.</p>
011		<p>Odd parity mode is active. D[9] is 0 since data 010_1010 has an odd number of 1's. The other data remains consistent.</p>

100	<table><tr><th>Name</th><th>Value</th></tr><tr><td>▶ baud[3:0]</td><td>1011</td></tr><tr><td>↳ TxRdy</td><td>0</td></tr><tr><td>↳ Tx</td><td>1</td></tr><tr><td>↳ clk</td><td>0</td></tr><tr><td>↳ reset</td><td>0</td></tr><tr><td>↳ load</td><td>1</td></tr><tr><td>↳ eight</td><td>1</td></tr><tr><td>↳ pen</td><td>0</td></tr><tr><td>↳ ohel</td><td>0</td></tr><tr><td>▶ out_port[7:0]</td><td>10100101</td></tr><tr><td>▶ shift_out[10:0]</td><td>11010010101</td></tr></table>	Name	Value	▶ baud[3:0]	1011	↳ TxRdy	0	↳ Tx	1	↳ clk	0	↳ reset	0	↳ load	1	↳ eight	1	↳ pen	0	↳ ohel	0	▶ out_port[7:0]	10100101	▶ shift_out[10:0]	11010010101	<p>Data is evaluated as an 8-bit value. D[9] matches with D[7] data and set to 1.</p> <p>No parity is enabled so the other data remains consistent.</p>
Name	Value																									
▶ baud[3:0]	1011																									
↳ TxRdy	0																									
↳ Tx	1																									
↳ clk	0																									
↳ reset	0																									
↳ load	1																									
↳ eight	1																									
↳ pen	0																									
↳ ohel	0																									
▶ out_port[7:0]	10100101																									
▶ shift_out[10:0]	11010010101																									
101	<table><tr><th>Name</th><th>Value</th></tr><tr><td>▶ baud[3:0]</td><td>1011</td></tr><tr><td>↳ TxRdy</td><td>0</td></tr><tr><td>↳ Tx</td><td>1</td></tr><tr><td>↳ clk</td><td>0</td></tr><tr><td>↳ reset</td><td>0</td></tr><tr><td>↳ load</td><td>1</td></tr><tr><td>↳ eight</td><td>1</td></tr><tr><td>↳ pen</td><td>0</td></tr><tr><td>↳ ohel</td><td>1</td></tr><tr><td>▶ out_port[7:0]</td><td>10100101</td></tr><tr><td>▶ shift_out[10:0]</td><td>11010010101</td></tr></table>	Name	Value	▶ baud[3:0]	1011	↳ TxRdy	0	↳ Tx	1	↳ clk	0	↳ reset	0	↳ load	1	↳ eight	1	↳ pen	0	↳ ohel	1	▶ out_port[7:0]	10100101	▶ shift_out[10:0]	11010010101	<p>Parity is not enabled. Output remains the same as above verification stimulus.</p>
Name	Value																									
▶ baud[3:0]	1011																									
↳ TxRdy	0																									
↳ Tx	1																									
↳ clk	0																									
↳ reset	0																									
↳ load	1																									
↳ eight	1																									
↳ pen	0																									
↳ ohel	1																									
▶ out_port[7:0]	10100101																									
▶ shift_out[10:0]	11010010101																									
110	<table><tr><th>Name</th><th>Value</th></tr><tr><td>▶ baud[3:0]</td><td>1011</td></tr><tr><td>↳ TxRdy</td><td>0</td></tr><tr><td>↳ Tx</td><td>1</td></tr><tr><td>↳ clk</td><td>0</td></tr><tr><td>↳ reset</td><td>0</td></tr><tr><td>↳ load</td><td>1</td></tr><tr><td>↳ eight</td><td>1</td></tr><tr><td>↳ pen</td><td>1</td></tr><tr><td>↳ ohel</td><td>0</td></tr><tr><td>▶ out_port[7:0]</td><td>10100101</td></tr><tr><td>▶ shift_out[10:0]</td><td>01010010101</td></tr></table>	Name	Value	▶ baud[3:0]	1011	↳ TxRdy	0	↳ Tx	1	↳ clk	0	↳ reset	0	↳ load	1	↳ eight	1	↳ pen	1	↳ ohel	0	▶ out_port[7:0]	10100101	▶ shift_out[10:0]	01010010101	<p>Even parity mode is active. D[10] is 0 since data 1010_0101 is in even parity mode.</p>
Name	Value																									
▶ baud[3:0]	1011																									
↳ TxRdy	0																									
↳ Tx	1																									
↳ clk	0																									
↳ reset	0																									
↳ load	1																									
↳ eight	1																									
↳ pen	1																									
↳ ohel	0																									
▶ out_port[7:0]	10100101																									
▶ shift_out[10:0]	01010010101																									
111	<table><tr><th>Name</th><th>Value</th></tr><tr><td>▶ baud[3:0]</td><td>1011</td></tr><tr><td>↳ TxRdy</td><td>0</td></tr><tr><td>↳ Tx</td><td>1</td></tr><tr><td>↳ clk</td><td>0</td></tr><tr><td>↳ reset</td><td>0</td></tr><tr><td>↳ load</td><td>1</td></tr><tr><td>↳ eight</td><td>1</td></tr><tr><td>↳ pen</td><td>1</td></tr><tr><td>↳ ohel</td><td>1</td></tr><tr><td>▶ out_port[7:0]</td><td>10100101</td></tr><tr><td>▶ shift_out[10:0]</td><td>11010010101</td></tr></table>	Name	Value	▶ baud[3:0]	1011	↳ TxRdy	0	↳ Tx	1	↳ clk	0	↳ reset	0	↳ load	1	↳ eight	1	↳ pen	1	↳ ohel	1	▶ out_port[7:0]	10100101	▶ shift_out[10:0]	11010010101	<p>Odd parity mode is active.</p> <p>Forces D[10] to be a 1 in order to make 1010_0101 have an odd number of 1's.</p>
Name	Value																									
▶ baud[3:0]	1011																									
↳ TxRdy	0																									
↳ Tx	1																									
↳ clk	0																									
↳ reset	0																									
↳ load	1																									
↳ eight	1																									
↳ pen	1																									
↳ ohel	1																									
▶ out_port[7:0]	10100101																									
▶ shift_out[10:0]	11010010101																									

## 7. RECEIVE ENGINE

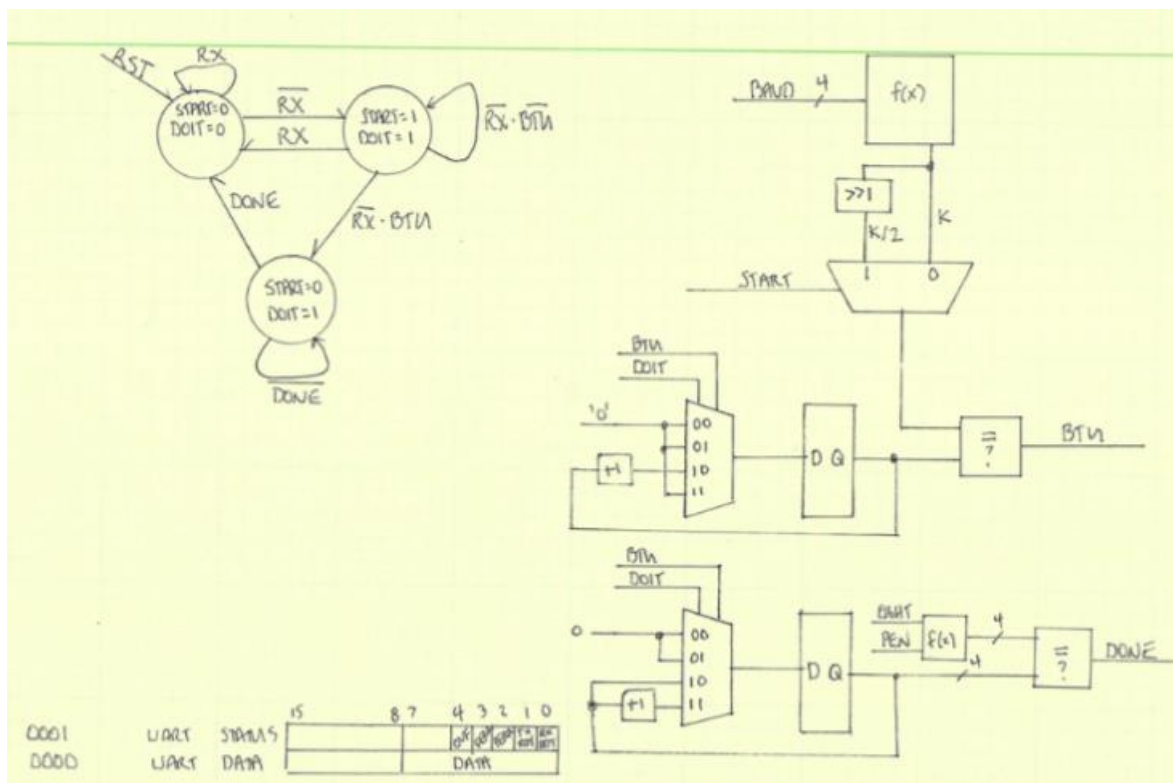
### 7.1 DESCRIPTION

The Receive(Rx) engine synchronizes the data collection with the Tx engine. The Rx engine is designed using multiple combinational logic blocks with the core being a Finite State Machine (FSM) and a shift register to collect serial data. The Rx engine continuously polls the Rx line for data anticipating a HIGH to LOW transition to indicate the arrival of a START bit. The data collected on the Rx line will range from nine-eleven bits with the first three values controlling the UART protocol settings.

### 7.2 RECEIVE ENGINE CONTROL

The purpose of the Receive Engine Control is to collect data from the Rx line and eliminate exchanging errors. The Rx engine control is designed using a finite state machine (FSM) to ensure the START bit remains low-active until the BTU signal is received to start collecting data at a specified bps (bits per second) until all bits are acquired. The BTU signal influences the bit counter and bit time counter logic blocks.

#### 7.2.1 BLOCK LEVEL DIAGRAM



#### 7.2.2 BIT TIME COUNTER

The Bit-time counter informs the Rx Engine the shifting rate of the current receiving data. The bit time counter is designed using a multiplexor register that increments a comparator variable.

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

The comparator evaluates the counted value to the selected baud rate controlling the speed of the data transmission. The signal (BTU) is HIGH active and notifies the shift register to shift out the current data when BTU signal is '1'. The BTU signal has two acceptable rates of bps; k/2 and k. The FSM uses the k/2 baud rate to detect the START bit while the succeeding data is processed at rate of k bps.

### 7.2.3 BIT COUNTER

The Bit Counter block is designed using multiple case-statements, a sequential logic block, and combinational logic blocks. One of the case-statements used controls the shifting of data when the BTU and DOIT signals are active. The Rx engine contains a decoder for determining the size of data to capture on the Rx line. The decoder outputs a total number of bits based on the values for eight<sup>2</sup> and parity enable.

eight	pen	# of bits to collect
0	0	9
0	1	10
1	0	10
1	1	11

### 7.2.4 FINITE STATE MACHINE

The core of the Rx engine control is composed of a finite state machine (FSM) that controls the data collection on the Rx line. The FSM continuously polls the Rx line while anticipating a HIGH to LOW transition. Upon recognition of a valid transition, indicating the arrival of a START bit, the FSM ensures the START bit is stable until the mid-bit time is reached which will enable the serial processing of data using the configured bit time. The FSM is designed to output the START and DOIT signals. The START signal indicates the Rx engine is searching for the START bit. The DOIT signal indicates the Rx engine is currently processing data from the Rx line.

## 7.3 RECEIVE ENGINE DATA PATH

The purpose of the Receive Engine data path is where the data will begin collection. The Rx engine data path is designed using a shift register and a remap function. The received data is analyzed to detect any errors during the exchange. There are four primary signals to observe from the Rx engine data path: RxRdy, PERR, FERR, and OVF.

<sup>2</sup> Referenced from page 6





Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

I	0		SB	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	X
I	I		SB	P	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

### Bit position after Remapping

eight	pen		9	8	7	6	5	4	3	2	1	0
0	0		0	0	SB	[6]	[5]	[4]	[3]	[2]	[1]	[0]
0	I		0	SB	P	[6]	[5]	[4]	[3]	[2]	[1]	[0]
I	0		0	SB	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
I	I		SB	P	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

### 7.3.4 RXRDY

The RxRdy signal indicates the Rx engine is ready to accept another stream of data. The RxRdy signal is generated using the RS flop<sup>3</sup>. When the DONE signal is generated, RxRdy will get set to a 1, informing the Rx engine is ready to collect data.

### 7.3.5 PARITY ERROR

Parity Error (PERR) logic block is designed so the Receive Engine can generate a parity based on the selected configurations and stream of data. When a parity bit is generated, it will evaluate with the detected data parity bit. If the both parity bits match, the PERR error signal will remain low until a mismatch of parity bits are detected. Parity Error is a weak form of data checking because parity essential only evaluates the number of ones in the data, instead of the bit position.

### 7.3.6 FRAMING ERROR

The Framing Error (FERR) logic block is designed to ensure that the received data within the bounds of the Rx engine determined data size. The FERR functionality utilizes the DONE signal and STOP bit value to detect a possible framing error. When both DONE and STOP are evaluated as asserted, the Rx engine will detect an error because the stop bit is analyzed as part of the data stream.

### 7.3.7 OVERFLOW ERROR

The Overflow Error (OVF) logic block is designed to enable the Rx engine to respond with an error when the FSM acquires more than 8-bits of data. Within the implementation, the RxRdy and DONE signals cannot be asserted at the same time to limit the data from exceeding 8-bits. If this dual assertion happens, the overflow bits of data are being processed in the Rx engine.

## 7.4 VERIFICATION

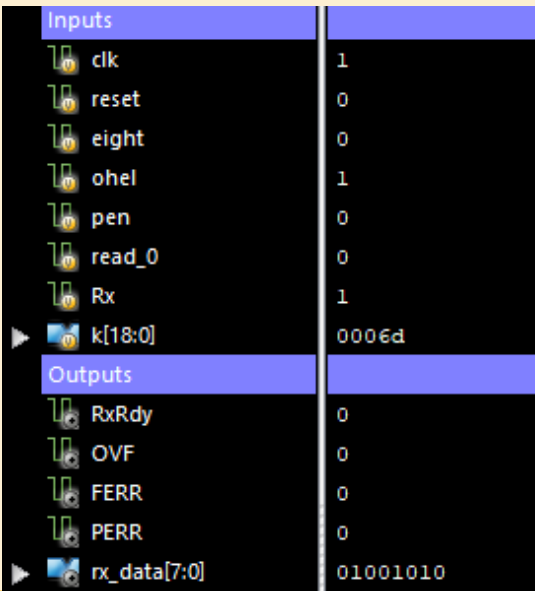
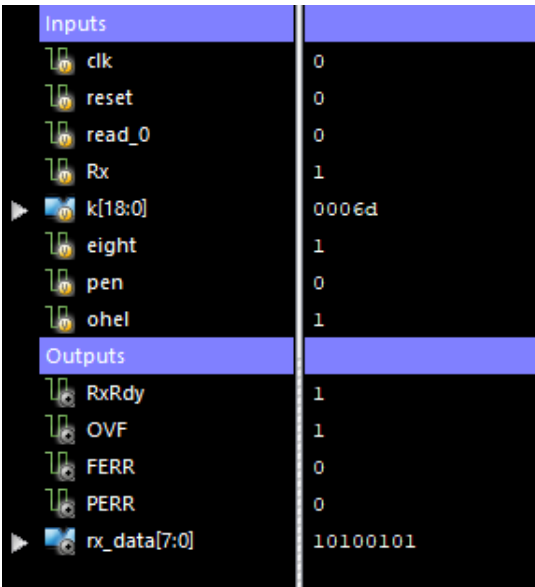
Verification is necessary to prove accountability of the many components within the design of the Rx Engine. For document version 2.0, the verification operation focuses on the **FSM, shift register, and Rx signal.**

<sup>3</sup> Referenced from page 15

Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

To verify the accuracy and proper functionality of the FSM and shift register, the Rx signal is set to one data bit per clock period, with a HIGH to Low transition for the START bit and LOW to HIGH transition STOP bit. Reset is asserted and deasserted after a fixed time to bring all registers to a stable-known state. To streamline the verification of the baud rates, the fastest baud rate of 4'b1011 is selected. All possible cases for the signals pen, ohel, and eight are enabled to verify the valid collected data. The sampling Rx data is set to a value of 8'hA5.

#### 7.4.1 RX ENGINE WAVEFORM VERIFICATION

Mode	Explanation	Waveform
<b>Eight not enabled</b>	When eight mode is not enabled, the eighth bit in on the rx_data stream is not evaluated to belong to the data stream.	
<b>Eight enabled</b>	When eight bit mode is enabled, the data stream is evaluated properly as seen on the rx_data line with value 1010_0101.	

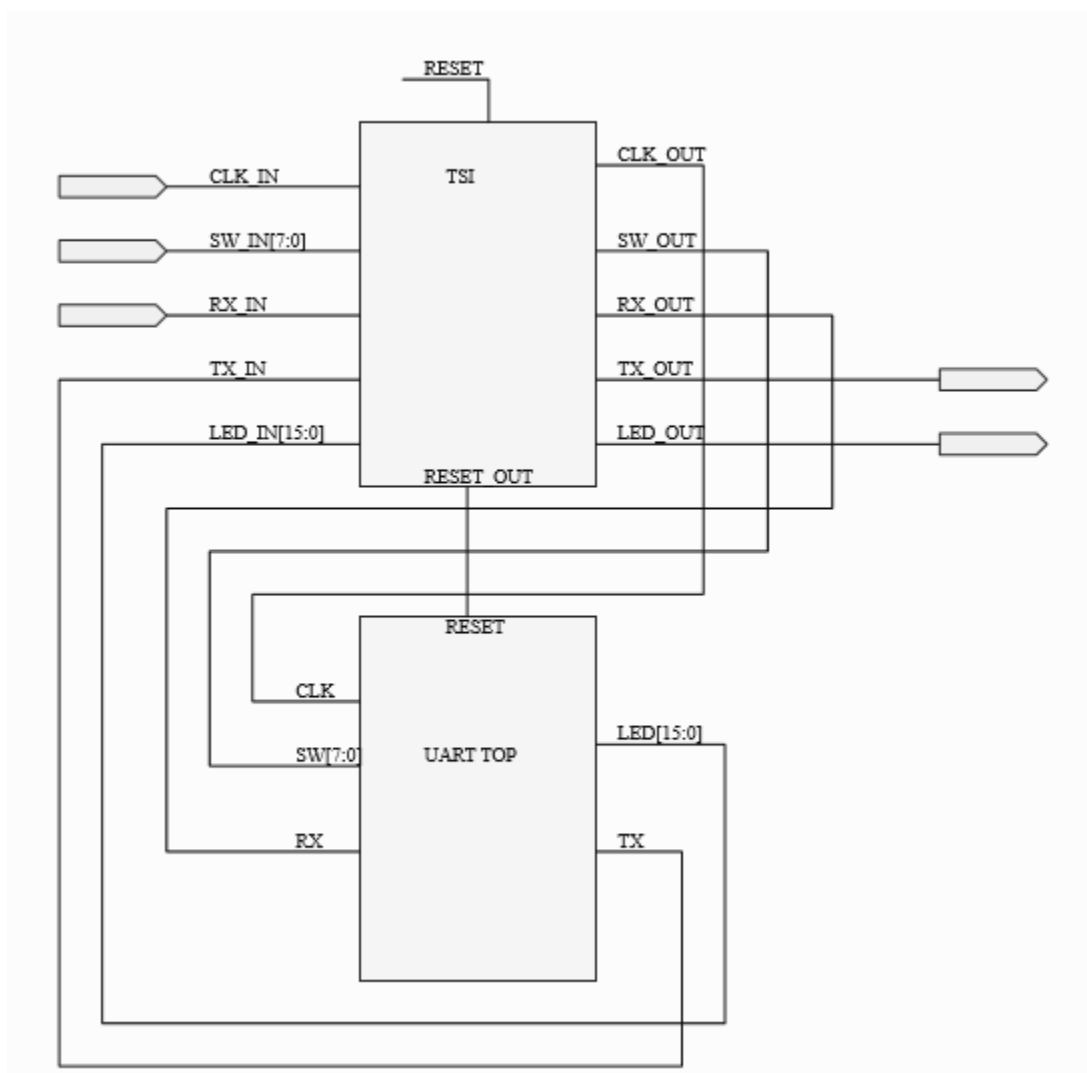
Written By: Zachery Takkesh	Date: December 1st, 2019	Revision: 3.0
--------------------------------	-----------------------------	------------------

## 8. TECHNOLOGY SPECIFIC INSTANTIATION

### 8.1 DESCRIPTION

Technology Specific Instantiation (TSI) contains all the references to the target libraries within the design. The TSI serves as a top wrapper of the entire design because all input/outputs must go through TSI before going to its destination module. Two buffers are used in the TSI module: IBUF and OBUF. The IBUF buffer gate is used to drive the inputs to the UART module. The OBUF buffer gate is used for driving the outputs of the UART module. The usage of buffers in the design meets the timing requirements in place.

### 8.2 BLOCK DIAGRAM



## 9. SOURCE CODE

Source Code is provided in a separate Document as a PDF.