2018



UART Full Duplex Protocol Chip Specification

Jose Sotelo

CECS 460

May 2, 2018

**Table of Contents**

**1 Introduction4**

1.1 Purpose4

2 Applicable Documents5

2.1 External Documents5

3 Requirements5

3.1.1 Performance Requirements5

3.1.2 Interface Requirements5

3.1.4 Physical Requirements5

3.1.5 Power Requirements5

3.1.6 Environmental Requirements5

4 Implementation6

4.1.1 UART Design Description6

4.1.2 UART Top Level Description7

4.1.3 UART Block Diagram7-8

4.1.4 UART Input/Output Interface Description9

4.1.5 Clocks9

4.1.6 Resets9

**5 Internally Acquired Blocks10**

5.1 Asynchronous-In Synchronous-Out(AISO)10

5.1.1 Description10

5.1.2 Block Diagram10

5.1.3 I/O Definition10

5.2 Full Duplex Universal Asynchronous Receiver Transmitter…………………………….10

5.2.1 Transmit Engine……………………………………………………………………10

5.2.2 Transmit Engine Block Diagram…………………………………………………..10

5.2.3 Receive Engine…………………………………………………………………….10

5.2.4 Receive Engine Block Diagram……………………………………………………10

**6 Externally Acquired Blocks10**

6.1 TramelBlaze Embedded Processor10

6.1.1 Description10

6.1.2 Block Diagram10

6.1.3 I/O Definition10-1

**7 Chip Verification11**

7.1 Block Name11

7.1.1 Testbench WaveForms11-12

**8 Software Code**

8.1 Tx\_Engine\_Top.v

8.2 Aiso.v

8.3 Tx\_Engine.v

7.3.1 SR\_Flop\_Txrdy.v

7.3.2 SR\_Flop.v

7.3.3 Load\_Reg.v

7.3.4 Parity\_Gen\_Dec.v

7.3.5 Shift\_Register.v

7.3.6 bit\_time\_counter.v

7.3.7 bit\_counter.v

7.4 PED.v

7.5 Parity\_Gen\_Dec\_tf.v

7.6 Shift\_Reg\_tf.v

7.7 Tx\_Engine\_tf.v

7.8 tx\_engine.tba

**Appendix A: Key Terms13**

1. Introduction

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

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

**1.1 Purpose**

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

**2. Applicable Documents:**

**2.1 External Documents**

PicoBlaze 8-bit Microcontroller User’s Guide

TramelBlaze

Programming TramelBlaze

**3. Requirements:**

**3.1 Performance Requirements**

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

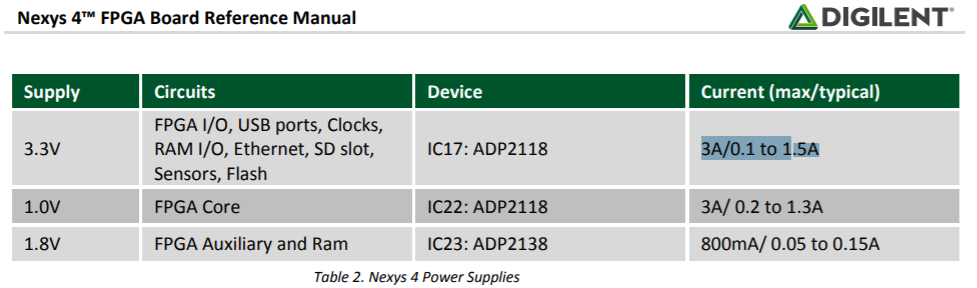
**3.1.2 Interface Requirements**

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

**3.1.4 Physical Requirements**

The highest baud rate the full duplex UART engine is capable to communicate via the serial terminal without breaking the data being sent is 460800.

**3.1.5 Power Requirements**



**3.1.6 Environmental Requirements**

**4. Implementation**

**4.1.1 Design Description**

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

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

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

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

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

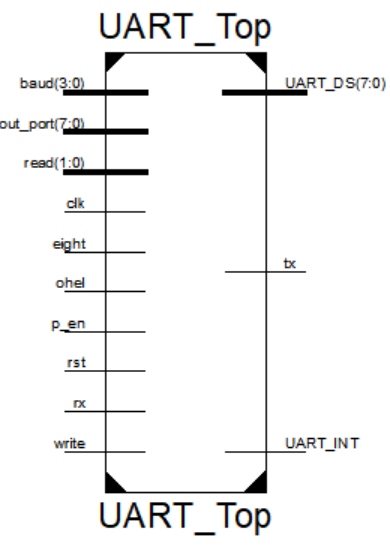
The design must also include two counters: 1) Count clocks to determine the bit time and 2) Count the bits to know when we are done. These two counters will let us know when we have waited a bit time and to know when all the bits have been transmitted. Lastly, a Baud Decoder circuit is created to generate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

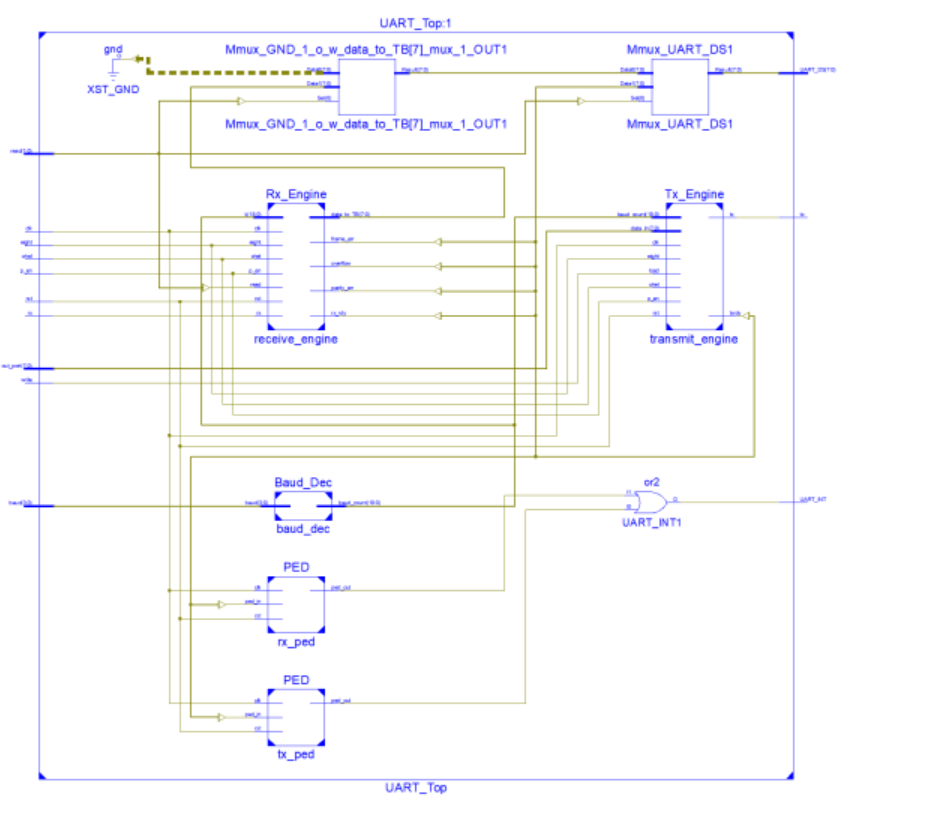
**4.1.2 UART Top Level Description**

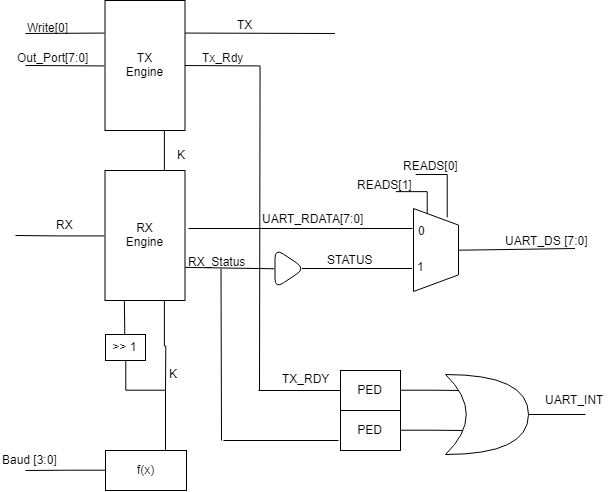
The UART is a serial, full duplex communication protocol which follows the RS-232 standard. The UART block is meant to model the capabilities, features, and the how to utilize the communication protocol.

**4.1.3 UART Block Diagram**

Top level diagram of the UART Engine







4.1.4 Input/Output Interface Description

Inputs

* Clk – 100MHz crystal oscillator
* Rst – Synchronous rest
* Out\_Port – 8-bit data coming from TramelBlaze
* Read – Read address 0001
* Write – Write address 0000
* Baud – 4-bit input used to select the baud rate

Pin Assignment: R13(SW8), U18(SW7), T18(SW6), R17(SW5)

* Rx – High to low transition
* Eight – 1-bit input to select [7:0] bits or [6:0] bits

Pin Assignment: R15(SW4)

* Parity\_En – 1: odd parity, 0: no parity

Pin Assignment: M13(SW3)

* OHEL – 1: odd parity, 0: even parity

Pin Assignment: L16(SW2)

Output

* UART\_DS – Status Flags
* Tx- Data that’s being shifted out from Transmit Engine
* UART\_Int –Interrupt that is feed into the TramelBlaze

4.1.5 Clocks

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

4.1.6 Resets

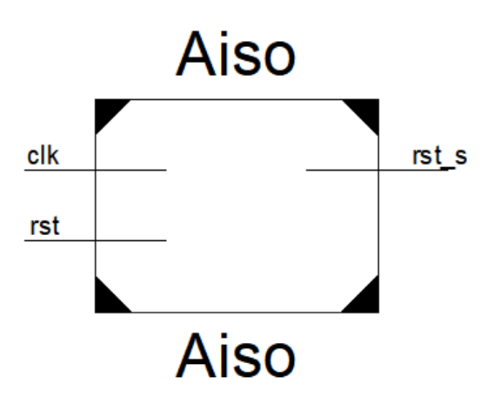
All the designs used a synchronous reset (AISO) to prevent metastability.

**5. Internally Acquired Blocks**

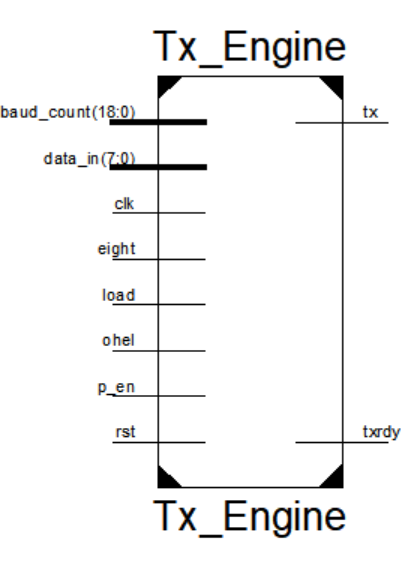
5.1 Asynchronous-In-Synchronous-Out (AISO)

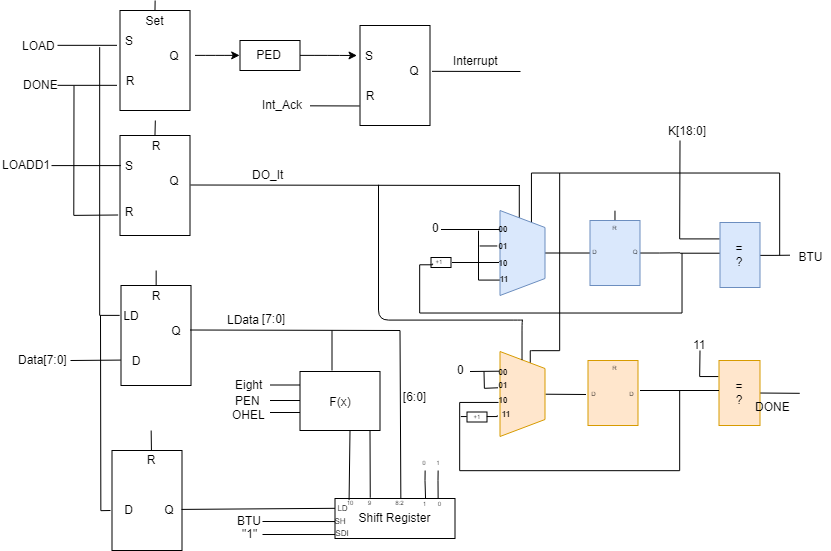
The Asynchronous-In-Synchronous-Out circuit is used to synchronize the reset of digital flip flops.

5.1.2 AISO Block Diagram

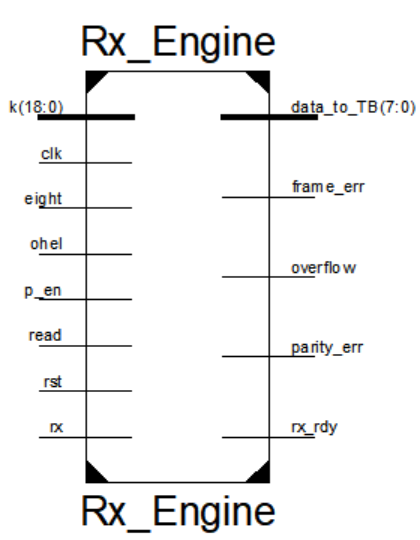


5.1.2 Full Duplex Universal Asynchronous Receiver Transmitter  
Block diagram of the Transmit Engine

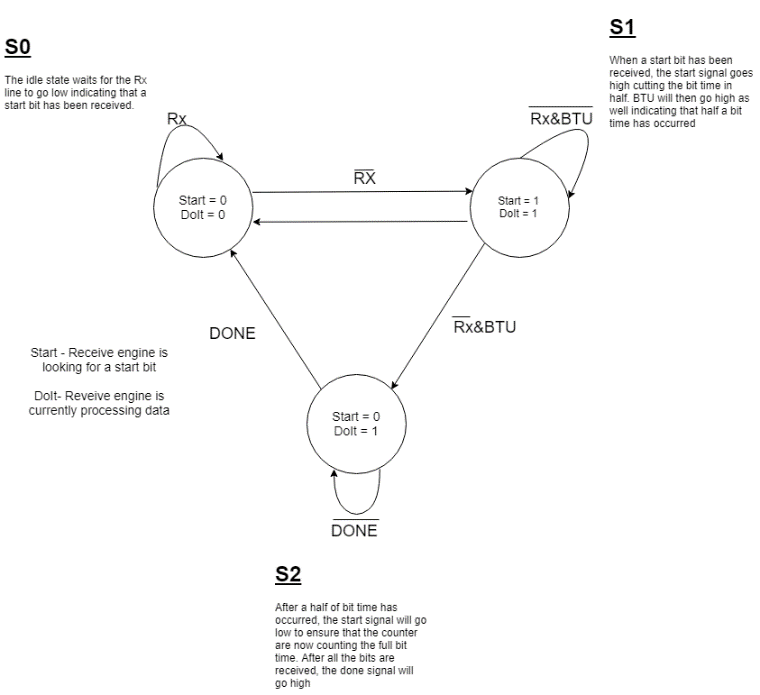




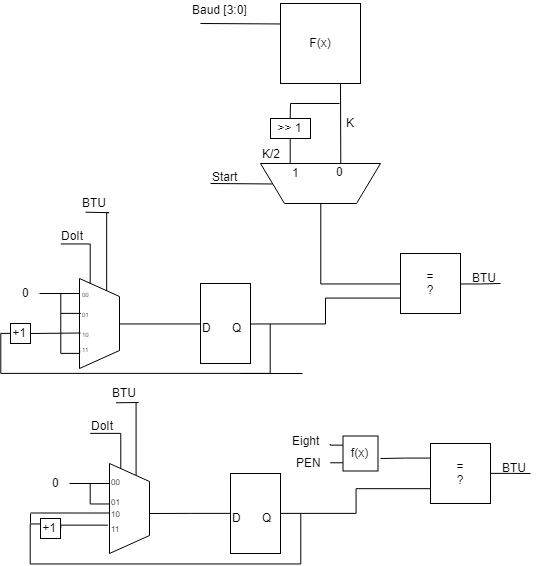
Block diagram of the Transmit Engine



Rx Finite State Machine



Rx Engine Control



Rx Engine Data Path

**6. Externally Acquired Blocks**

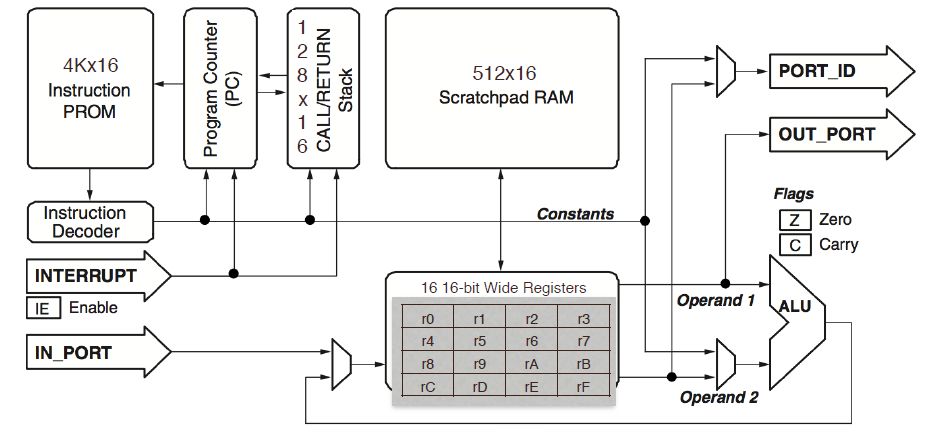
5.1 Block Name

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

5.1.1 Description

The TramelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze. The instruction set is the PicoBlaze instruction set. It contains three memories Instructions Rom, Call/Return Stack and Scratchpad RAM. The TramelBlaze’s I/O are a 16-bit In\_Port, Interrupt, Reset, Clk, a 16-bit Out\_Port, 16-bit Port\_Id output, Read\_Strobe output, Write\_Strobe output and interrupt acknowledge output. When implementing the TramelBlaze into your design, you must generate the tb\_rom and load the coe file into RAM. A Python assembler is used to generate assembly code. We will be using assembly code to output to the serial terminal and keep track of the line count.

5.1.2 Block Diagram



The TramelBlaze MCU block diagram

5.1.3 I/O Definitions

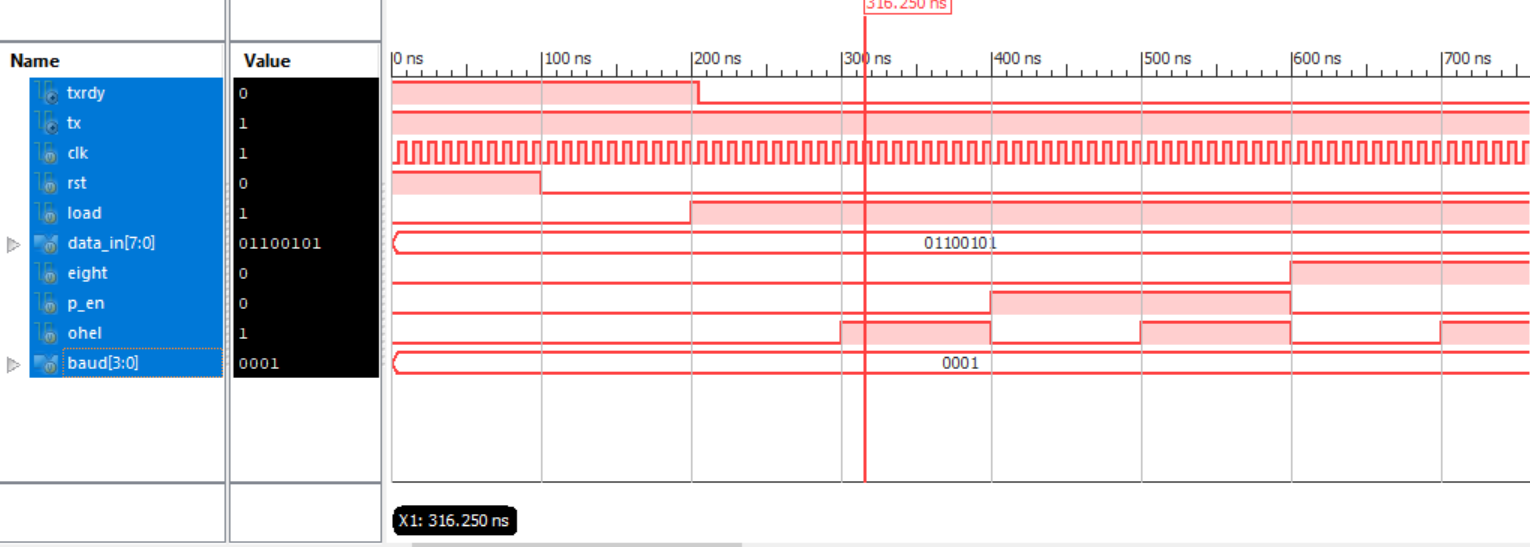
The I/O ports extend the TramelBlaze MCU’s capabilities and allow the MCU to connect custom peripherals or to FPGA. It contains a 16-bit In\_Port input that is used to write to the scratchpad memory. It allows to use 16 16-byte-wide registers. It also handles the interrupt input used to let the MCU that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

**6. Chip Verification**

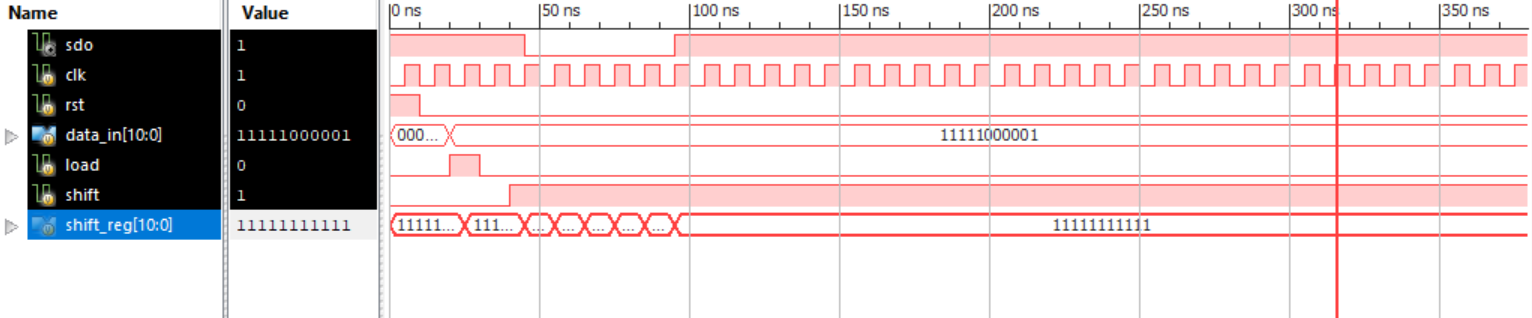
6.1 Block Names

* Transmit Engine TestFixture
* Shift Register TestFixture
* Parity Decoder TestFixtue

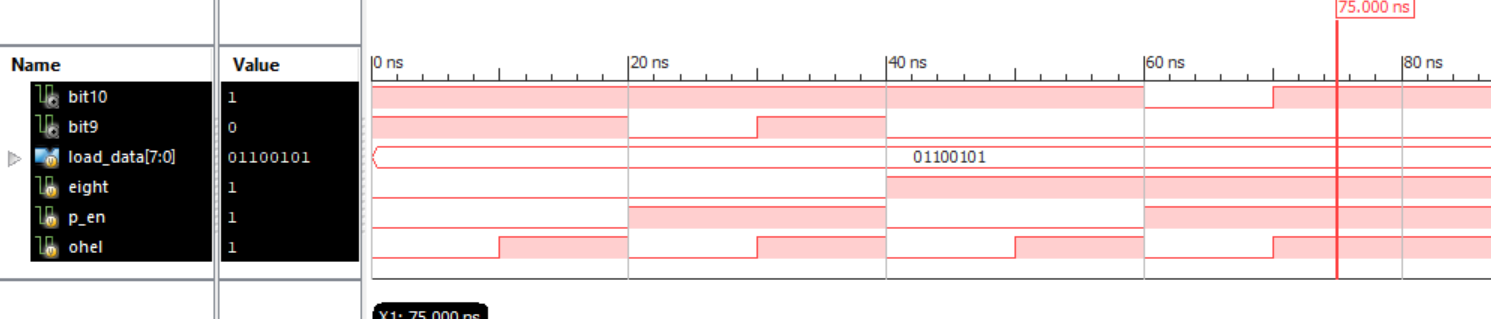
6.1 Testbench Waveforms



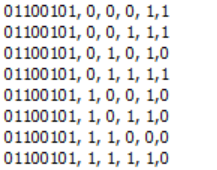
Transmit Engine



Shift Register



Parity Decoder



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

**7. Software Code**

**Appendix A: Key Terms**

**Start Bit**

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

**Data Frame/Packet**

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

**Parity**

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

**Stop Bit**

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

**Baud Rate**

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

**Bit time**

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

*Illustration of Data Transmission*

