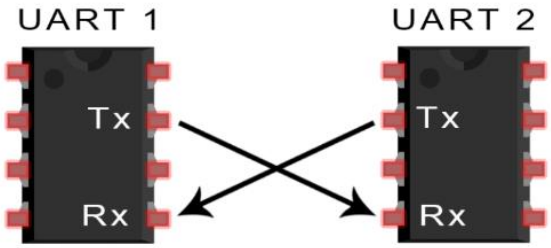
Mini Project

Design of a UART

INTRODUCTION TO UART COMMUNICATION

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:

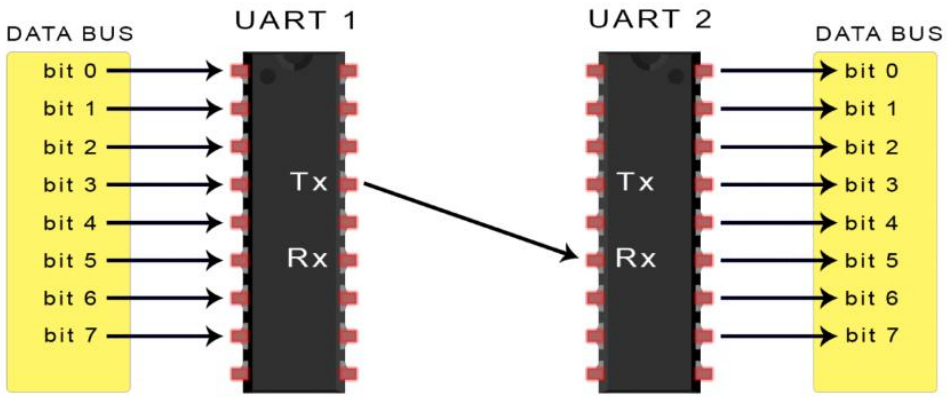


UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

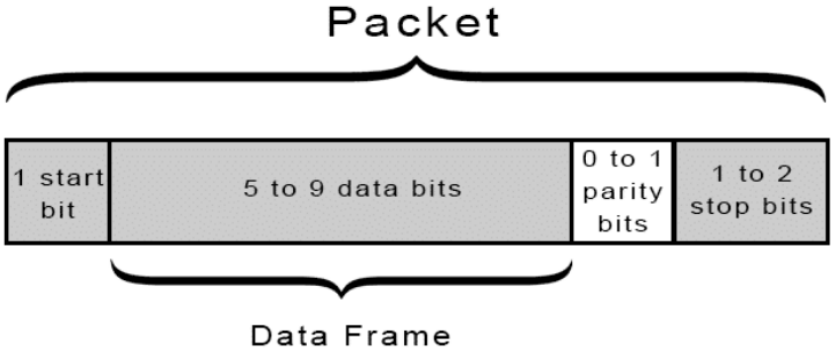
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate*. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

HOW UART WORKS

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end:



UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



START BIT

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to 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.

DATA FRAME

The data frame contains the actual data being transferred. It can be 5 bits up 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 most cases, the data is sent with the least significant bit first.

PARITY

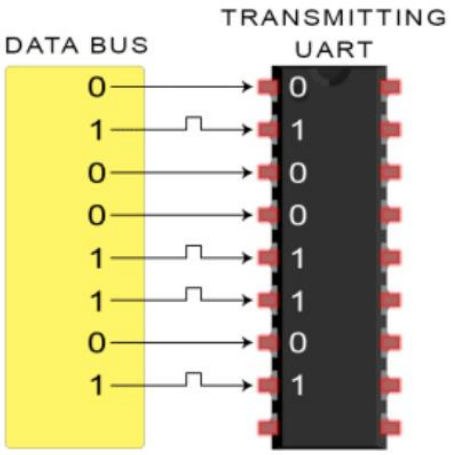
Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long distance data transfers. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 bits in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bits in the data frame should total to an odd number. When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

STOP BITS

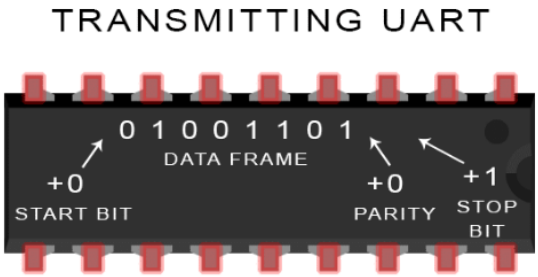
To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

STEPS OF UART TRANSMISSION

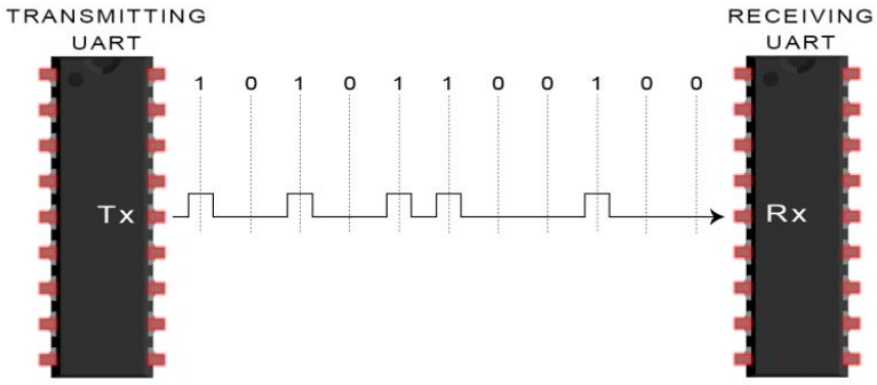
1. **The transmitting UART receives data in parallel from the data bus:**



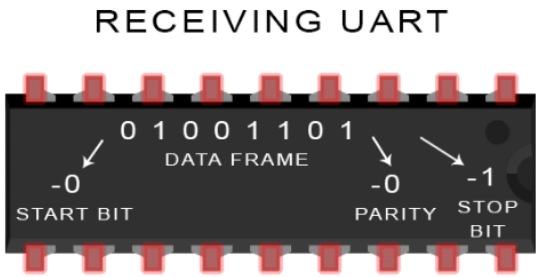
2. **The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:**



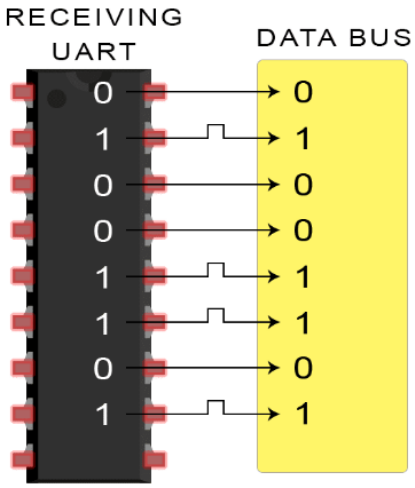
3. **The entire packet is sent serially from the transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate:**



4. **The receiving UART discards the start bit, parity bit, and stop bit from the data frame:**



5. **The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end:**



Overview

- PC sends and receives characters via USB cable.
  - Uses a terminal program (such as HyperTerminal or Tera Term) and a Virtual Communication Port
- USB-to-UART bridge converts the USB protocol to RS-232 protocol.

Basic Description

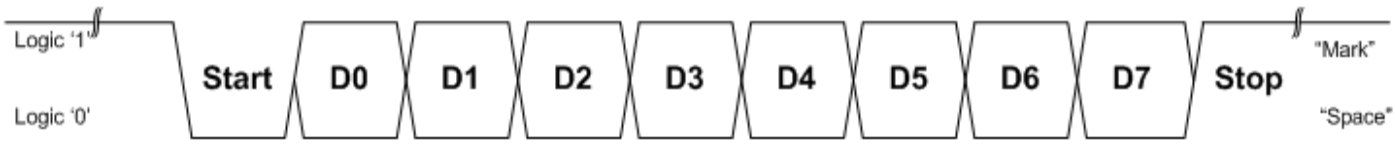
- This design *records* specific information via RS-232 serial communication.
- After data has been stored, it can be retrieved via the RS-232 communications channel or *played* out via a bank of LEDs.
- Receives RS-232 serial data at 115200 baud (no parity, 8 data bits, no handshaking)
  - Handled in *uart_rx,* which is simple state machine and an over sampler.

Introduction to RS-232-C

- RS-232-C is a standard issued by the Electronics Industry Association (EIA).
- Used since the late 1960s for communication between Data Terminal Equipment (DTE) and Data Communication Equipment (DCE).
- Allows communication between DTEs and DCEs over a short range at relatively high speeds*
- While not technically covered by the standard, commonly used transmission rates, framing, and character coding is usually considered part of RS-232.
- Still available on most modern computers.
- Simply referred to as the *serial port* or *COM port.*
- USB to UART bridge for PCs that lack an RS-232 serial port.

RS-232-C Protocol

- Data communication is character oriented and serially transmitted.
  - A character is typically an 8-bit byte
  - Characters are usually encoded, using the American Standard Code for Information Interchange (ASCII)
  - Framing is usually accomplished, using a single START bit before the character and a single STOP bit after the character.
  - The character is sent with the least-significant bit first.
  - The line is driven high (or *Mark*) when idle.
  - A START bit drives the line low (or *Space*) and the STOP bit is a Mark



RS-232-C Timing

- RS-232-C communication is asynchronous.

  No clock is sent along with the data.

- The equipment on both ends must be set to the same baud rate.

  The baud rate is the number of symbols per second sent on the line Each symbol is one bit; baud rate is the number of bits per second.

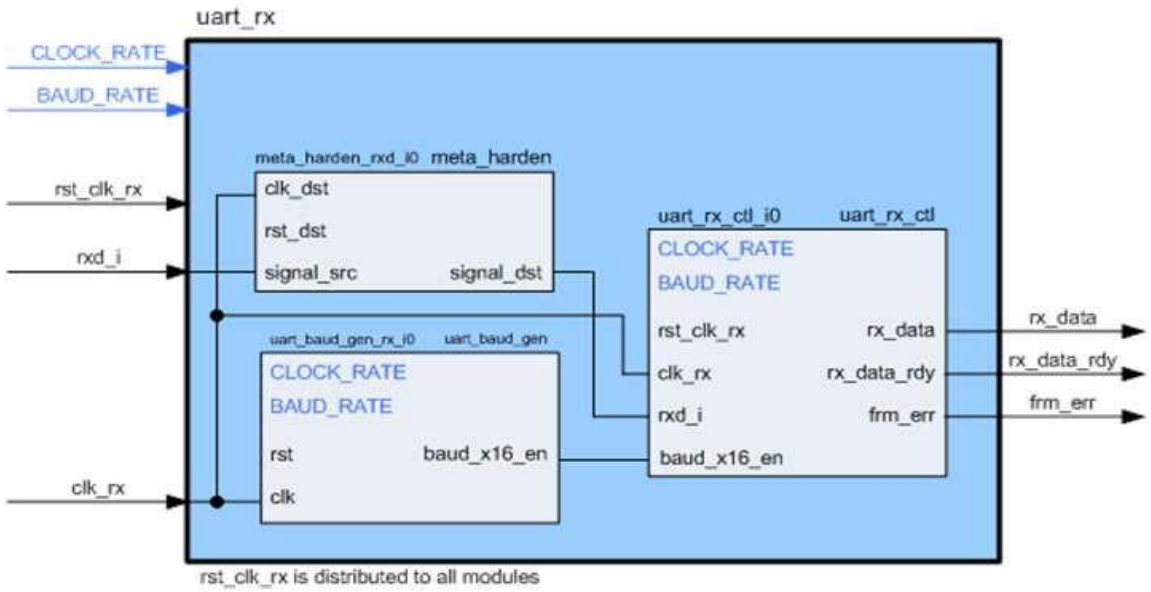
- Each bit (START, DATA, STOP) is sent on the line for the same amount of time.

  One bit period is 1 / BAUD_RATE

- The transmitter and receiver have independent time references.

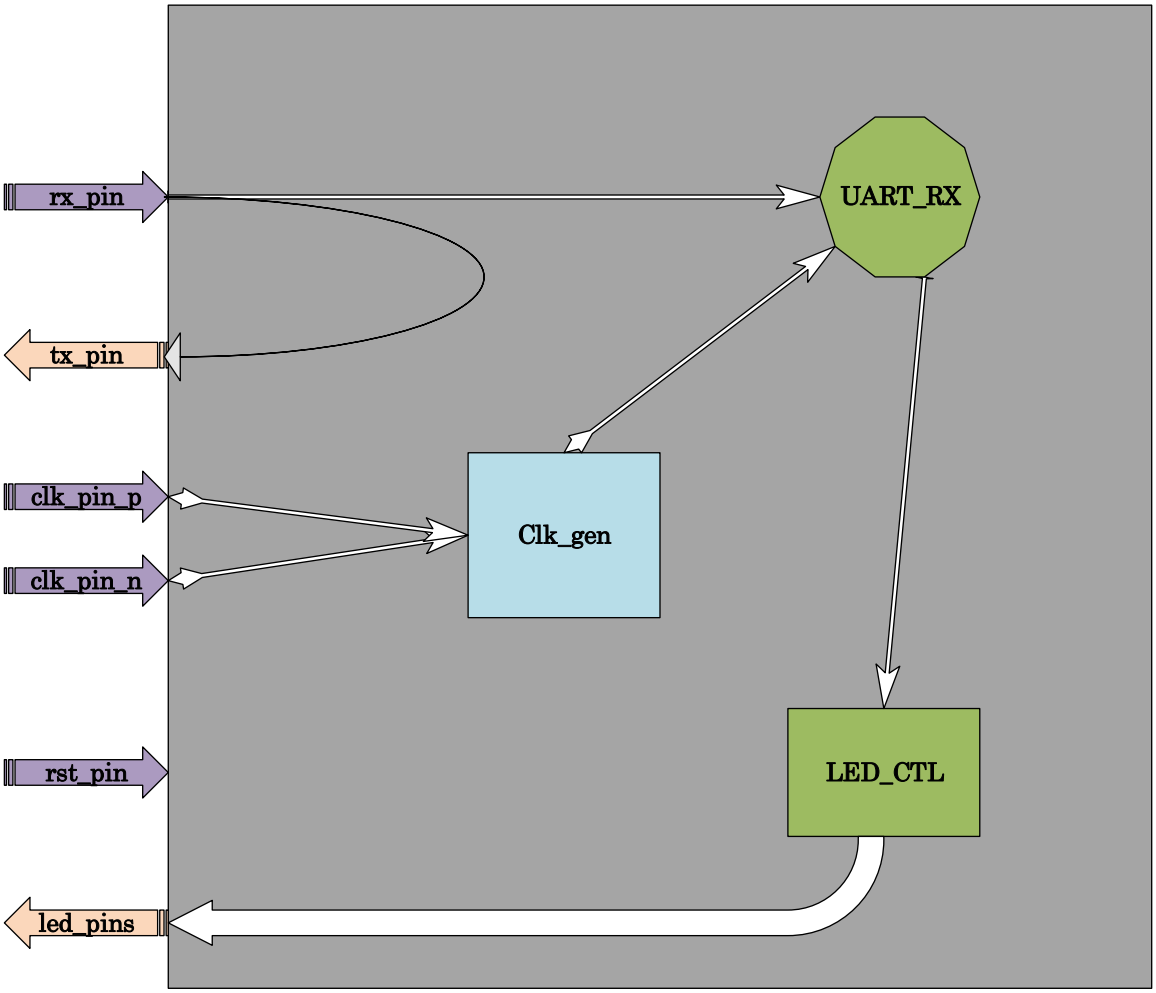
  The standard allows for some difference between the local references.

uart_rx Module



What to implement

You must follow the following Diagram:



Modules Description

Module	Description
UART_RX	Contains two submodules: • Rx receiver control of UART data from pc. • uart_baud_gen to generate baud enable. • Synchronizer(meta_harden).
Clk_gen	Generate clock to our system from differential clock pins on the board.

LED_CTL	Control the leds according on the coming numbers-> when 0 , no led turning on when 1 , 1 led turning on when 2 , 2 leds turning on when 3 , 3 leds turning on when 4 or greater than 4, all leds turning on

1. Using RTL Verilog, implement the above diagram using synchronous sequential and combinational logic.
2. Write a testbench that verifies the RTL code.
3. Compile and simulate your design.
4. Connect the required pins in .ucf file.