**ReadMe\_Uart\_tx - ChatGPT**

Libraries and use statements: The code includes three IEEE libraries: "ieee.std\_logic\_1164.all" for working with standard logic types, "ieee.std\_logic\_arith.all" for arithmetic operations on "std\_logic\_vector" signals, and "ieee.std\_logic\_unsigned.all" for unsigned arithmetic operations on "std\_logic\_vector" signals.  
  
Entity declaration: The entity "Uart\_tx\_Rom" defines the input and output ports of the UART transmitter module. It has four ports: "resetn" (active low reset), "sysclk" (system clock), "start\_triger" (start trigger signal), and "uart\_tx\_triger" (UART transmit trigger signal).  
  
Architecture declaration: The architecture ab specifies the internal implementation of the UART transmitter.  
  
Component declaration: The component "Rom\_X" is declared within the architecture. It represents a ROM module that is used to store data.  
  
Signal and variable declarations: Several signals and variables are declared within the architecture:  
  
"state\_tx" is of type state (an enumerated type) and represents the current state of the UART transmitter.

"signal\_A\_q" and "signal\_A\_q\_not" are signals that store the current and inverted values of the baud clock signal, respectively.

"sig\_arising\_edge" is a signal that represents the rising edge of the baud clock signal.

"sig\_bit" is a signal that represents the current bit being transmitted.

"sig\_baud\_clk" is a signal that stores the generated baud clock signal.

"sig\_address" is a signal that holds the current ROM address being accessed.

"sig\_rom\_data\_out" is a signal that stores the data read from the ROM.

"sig\_byte" is a signal that stores the current byte being transmitted.

"sig\_packet" is a signal that holds the complete packet to be transmitted.

ROM instantiation: An instance of the "Rom\_X" component is instantiated as dut within the architecture. It connects the ROM module to the UART transmitter and provides the data to be transmitted.  
  
Baud clock generation: The "baud\_clock" process generates a baud clock signal ("sig\_baud\_clk") used for synchronizing the data transmission. It increments a counter ("cnt\_baud") on every rising edge of the system clock ("sysclk"). When the counter reaches a specific value (651 in this case), it toggles the baud clock signal and resets the counter.

Rising edge detection: The "rising\_edg process" detects the rising edge of the baud clock signal ("sig\_baud\_clk"). It assigns the current value of "sig\_baud\_clk" to "signal\_A\_q" and the inverted value of "sig\_baud\_clk" to "signal\_A\_q\_not" on each rising edge of the system clock ("sysclk").  
  
Transmission process: The transmission process implements the main functionality of the UART transmitter using a finite-state machine. It operates based on the current state ("state\_tx") of the transmitter.

* The initial state (s0) waits for the "start\_triger" signal to transition to high to initiate the transmission process.
* In states s1 and s2, the transmitter waits for the "start\_triger" signal to transition to low and introduces a delay by incrementing "sig\_cntr" until it reaches 2.
* In state s3, the "sig\_address" is incremented to access the next address in the ROM.
* In states s4 and s5, the transmitter moves to the next state and reads the data
* State s6 introduces a delay by incrementing "sig\_cntr" until it reaches 5.
* In state s7, the transmitter constructs the packet by concatenating the start bit, data byte ("sig\_byte"), and end bits into "sig\_packet".
* In state s8, the transmitter moves to the next state.
* In state s9, the transmitter shifts the bits of "sig\_packet" and sends them out one by one based on the rising edge of "signal\_A\_q". It also increments "sig\_cntr" to keep track of the number of bits transmitted. Once "sig\_cntr" reaches 12, the transmitter moves to state s10.
* In state s10, the transmitter checks if the entire transmission is complete by comparing "sig\_address" with "000000". If the comparison is true, indicating that all ROM addresses have been transmitted, the transmitter returns to state s0 to wait for the next "start\_triger". Otherwise, it goes back to state s2 to continue the transmission process.

UART output: Finally, the "uart\_tx\_triger" output is assigned the value of "sig\_bit", representing the current bit being transmitted.

Overall, the code implements a basic UART transmitter using an FSM approach. It uses a ROM module to store the data to be transmitted and generates a baud clock signal to synchronize the transmission at a specific rate. The transmitter follows a specific sequence of states to read data from the ROM, construct packets, and transmit them serially with appropriate start and stop bits.

**Introduction Uart**

UART (Universal Asynchronous Receiver Transmitter) is a communication protocol used for serial communication between two devices. It is commonly used for transmitting and receiving data between a microcontroller and other devices such as sensors, displays, and other microcontrollers.

In UART, data is transmitted serially bit-by-bit over a single communication line. The transmitter sends data in a predefined format, which consists of a start bit, the data bits (typically 8 bits), an optional parity bit for error checking, and a stop bit. The start bit is always a low voltage level and signals the receiver to prepare to receive data. The data bits represent the actual data being transmitted and can have a value of 0 or 1. The optional parity bit is used for error checking and can be set to odd, even, or no parity. The stop bit is always a high voltage level and signals the end of the transmission.

On the receiving end, the UART module detects the start bit and begins sampling the data at a predefined baud rate (bits per second). Once all the data bits have been received, the parity bit (if used) is checked to ensure the data is error-free. Finally, the stop bit is detected, and the UART module signals the microcontroller that the data is ready to be processed.

Overall, UART is a simple and efficient communication protocol that is widely used in embedded systems and other applications. Its simplicity and low overhead make it an attractive option for many communication tasks.