**ReadMe\_Uart\_rx - ChatGPT**

**Libraries and Entity:**  
  
The code includes the IEEE libraries "ieee.std\_logic\_1164.all", "ieee.std\_logic\_arith.all", and "ieee.std\_logic\_unsigned.all."

These libraries provide standard types and operations for working with signals in VHDL.

The entity declaration defines the input and output ports of the UART receiver.

"resetn": Asynchronous reset signal.

"sysclk": System clock input.

"toggle": Input signal used for addressing the RAM.

"detected\_bit": Input signal representing the received data bit.

"wr\_ram": Output signal for controlling the RAM write operation.

"ram\_address": Output signal for addressing the RAM.

"detected\_byte": Output signal for the received byte of data.

**Architecture:**  
  
The architecture is named ab and defines the internal signals and processes of the UART receiver.

Signals:  
"state\_rx": A signal of type state, which is an enumeration representing the different states of the receiver.

"sig\_baudx32": A clock signal that operates at 32 times the baud rate of 38,400 bps. It toggles every 20 "sysclk" cycles.

"sig\_bit": A signal of type "std\_logic\_vector(2 downto 0)" used for debouncing the received data bit.

"sig\_bouncer\_bit": A debounced version of the received data bit.

"signal\_A\_q" and "signal\_A\_q\_not": Signals used to create a rising edge detector for "sig\_baudx32."

"sig\_araising\_edge": A signal that goes high on the rising edge of "sig\_baudx32", indicating the start of a bit.

"sig\_data\_bit": A signal representing the current received data bit.

"sig\_wr\_ram": A signal used to control the write operation of the RAM.

"sig\_ram\_address": A signal used to address the RAM.

"sig\_cnt\_address": A counter signal used to increment the RAM address.

"sig\_detected\_byte": A signal used to store the received byte of data.

**Processes:**

a. "baud\_rate\_clk":  
  
This process generates the "sig\_baudx32" clock signal with a frequency 32 times higher than the desired baud rate.

The "var\_cntr" variable is incremented on each rising edge of sysclk, and when it reaches 20, "sig\_baudx32" is toggled.

b. "araising\_edge":  
  
This process creates a rising edge detector for "sig\_baudx32" using the "signal\_A\_q" and "signal\_A\_q\_not" signals.

When sysclk has a rising edge, the "signal\_A\_q" and "signal\_A\_q\_not" signals are updated.

c. "debouncer":  
  
This process debounces the received data bit by comparing its value over three consecutive cycles.

The "sig\_bit" signal shifts the received bit value, and if the three bits are the same, "sig\_bouncer\_bit" is updated.

d. "main\_rx:"  
  
This is the main process that controls the behavior of the UART receiver.

It operates based on the current state, "state\_rx", which represents the current state of the receiver.

The process transitions between different states to receive and process the incoming data.

* In s0 state, if "sig\_bouncer\_bit" is high, it resets the "var\_bit\_cntr" and increments the   
  "var\_trns\_cntr". If "var\_trns\_cntr" reaches the desired value (500,000 in this case), it resets the "sig\_ram\_address" to all zeros. If "sig\_bouncer\_bit" is low, it transitions to s1 state.
* In s1 state: If "sig\_araising\_edge" is high, it increments the "var\_clk\_cntr". When "var\_clk\_cntr" reaches 16, it assigns the value of "sig\_bouncer\_bit" to "sig\_data\_bit". When "var\_clk\_cntr" reaches 32, it resets "var\_clk\_cntr" and transitions to s2 state.
* In s2 state: If "sig\_data\_bit" is low, it transitions to s3 state and assigns the inverted value of toggle concatenated with "sig\_cnt\_address" to "sig\_ram\_address". If "sig\_data\_bit" is high, it transitions back to s0 state.
* In s3 state: If "sig\_araising\_edge" is high, it increments "var\_clk\_cntr". When "var\_clk\_cntr" reaches 16, it assigns the value of "sig\_bouncer\_bit" to "sig\_data\_bit" and increments "var\_bit\_cntr". When "var\_clk\_cntr" reaches 32, it resets "var\_clk\_cntr" and appends "sig\_data\_bit" to the "sig\_detected\_byte." If "var\_bit\_cntr" reaches 8, it transitions to s4 state.
* In s4 state: It assigns the value of "sig\_detected\_byte" to the output "detected\_byte". It increments "sig\_cnt\_address" for the next RAM address. It transitions to s5 state.
* In s5 state: It sets "sig\_wr\_ram" high to enable the RAM write operation. It transitions back to s0 state. For any other state, it transitions back to s0 state.

Output Ports:

The "signals sig\_wr\_ram", "sig\_ram\_address", and "sig\_detected\_byte" are assigned to the corresponding output ports of the entity "Uart\_rx".

These ports ("wr\_ram", "ram\_address", and "detected\_byte") are used to communicate with external components, such as the RAM, to perform the desired operations.

Note: The provided code snippet is a partial implementation of the UART receiver. It lacks the instantiation of components and the actual RAM module or interface. To complete the implementation, you would need to instantiate the necessary components and define the RAM module or interface to handle the write operation ("sig\_wr\_ram") and RAM address ("sig\_ram\_address") based on your specific requirements.

**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.