**1.Introduction**

**1.1 Background**

In the landscape of digital communications, the Universal Asynchronous Receiver/Transmitter (UART) is a key technology for serial data transmission, vital in a variety of applications from simple device communication to sophisticated industrial control systems. The FPGA (Field-Programmable Gate Array), known for its flexibility and performance efficiency, provides an ideal platform for the implementation and exploration of UART protocols.

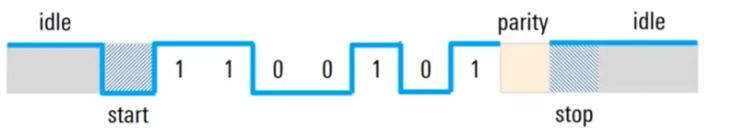


Figure 1. structure of UART frame

**1.2 System Overview**

The UART transmitter system explored in this task is an assembly of several critical modules, each contributing to the system’s functionality:

* Controller: Manages the overall operation and coordination of the system.
* Shift Register: Responsible for holding and serially transmitting data bits.
* Baud Rate Generator: Defines the transmission speed or baud rate of the system.
* Bit Counter: Tracks the number of bits transmitted, ensuring complete data frame transmission.
* Parity Generator: Produces a parity bit for error checking in the data transmission.
* Seven Segment Decoder: Displays the data input in a human-readable format, aiding in debugging and system

**1.3 Aim Of The Task**

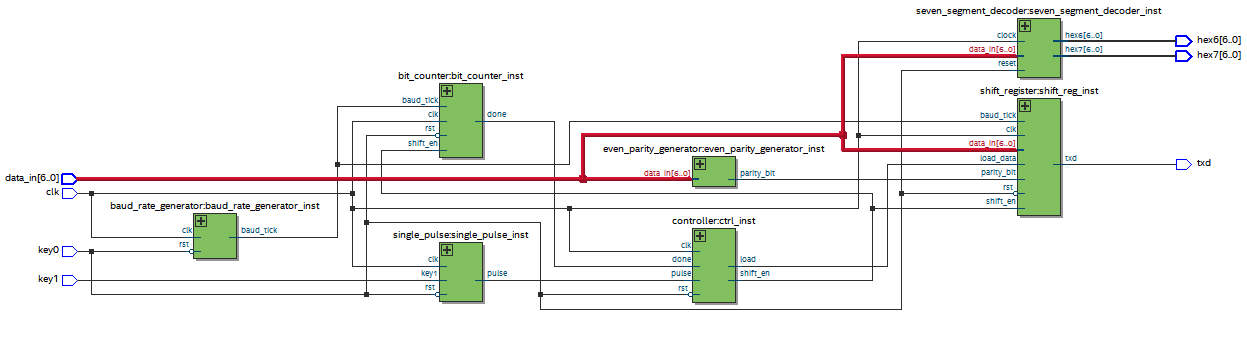
.The purpose of this task is to design and implement a UART transmitter system on an FPGA, involving the creation and integration of various modules like a Controller, Shift Register, Baud Rate Generator, Bit Counter, Even Parity Generator, and a Seven Segment Decoder. After coding these modules in Verilog and assembling them into a functional system, the task extends to conducting simulations and practical testing. This testing phase involves using a tool like Putty to establish a serial communication link with the FPGA, allowing for the real-world validation of the UART transmitter by observing and verifying the data transmitted in ASCII format. This comprehensive process ensures the system's functionality and effectiveness in actual operating condition.

**2.Module Design**

**2.1 The Architecture Of The UART Transmitter**

The architecture of the UART transmitter system designed for FPGA implementation comprises several interconnected modules, each serving a distinct role in the process of serial data transmission:

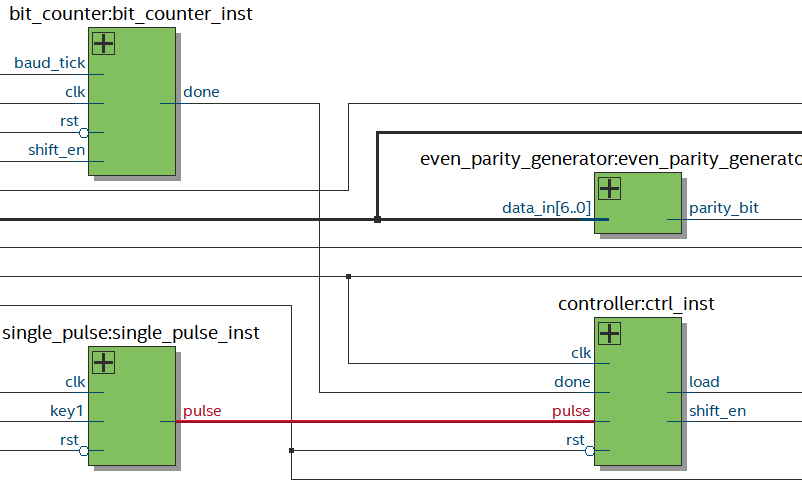
* Controller: The central unit that orchestrates the overall operation. It manages the initiation and control of data transmission, coordinating the data flow and timing among the modules.
* Shift Register: Responsible for storing and serially shifting out the data bits, including the start bit, data bits, parity bit, and stop bit, following the UART protocol.
* Baud Rate Generator Generates the timing signals (baud rate) to control the speed of data transmission, ensuring consistent and pre-set timing for transmitting data bits.
* Bit Counter: Tracks the number of bits transmitted. It signals the completion of a UART frame transmission after the predetermined number of bits (including start, data, parity, and stop bits) is reached.
* Even Parity Generator: Calculates the parity bit for error checking, ensuring an even number of '1' bits in the frame, including the parity bit.
* Seven Segment Decoder: Converts the digital data into a format for a seven-segment display, providing a visual representation of the transmitted data, useful for monitoring and debugging.
* Single Pulse: This module generates a single pulse signal, typically used to trigger the start of data transmission. It acts upon a specific user action or condition, such as pressing a button, to initiate the transmission process.



**Figure2.RTL review of UART Transmitter**

**2.1.2 Structure Of Controller**

The controller in a UART transmitter system plays a pivotal role in managing the entire transmission process, from the moment it receives the initial signal to the point where it concludes operations upon receiving a 'done' signal. Here's a detailed description of this process:



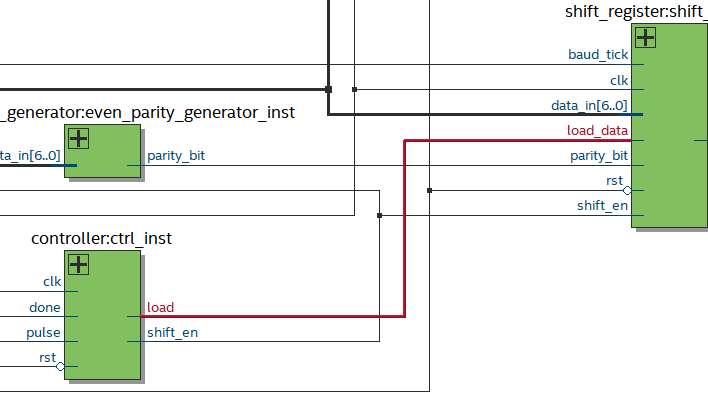
**Receiving the Pulse Signal:**

* The controller's operation begins when it receives a trigger, often generated by the Single Pulse module in response to an event like a button press.
* This trigger indicates that there's data ready to be transmitted

**Figure3 .RTL view of Pulse Signal**

**Loading Data into the Shift Register:**

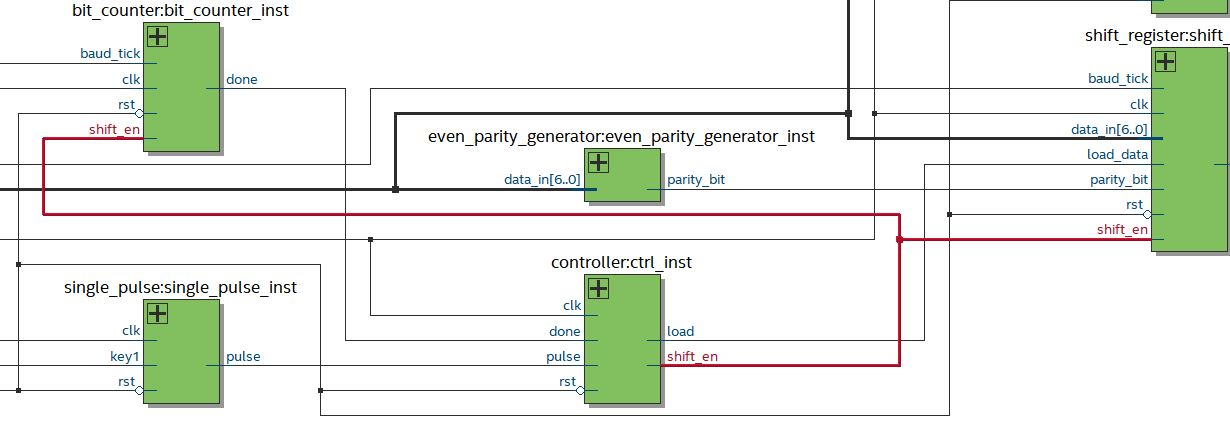
* Upon receiving the trigger, the controller activates the 'load' signal.
* This signal instructs the Shift Register to load the data to be transmitted. The data typically includes a start bit, the actual data bits, a parity bit for error checking, and a stop bit.



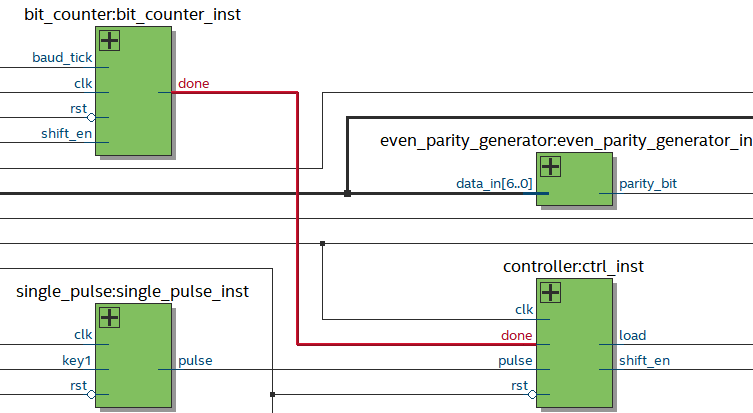
**Figure 4 .RTL view of load signal**

**Enabling Data Transmission:**

* The controller then generates a 'shift\_en' (shift enable) signal. This signal enables the Shift Register to start shifting out data bits and bit counter.



**Figure 5 .RTL view of shift\_en Signal**



**Receiving the 'Done' Signal**:

* Once the entire data frame is transmitted, the Bit Counter sends a 'done' signal to the controller.
* This signal indicates that the transmission of the current data frame is complete.

**Figure6 .RTL view of done signal**

**3. Description Of Each Module Design**

**3.1 Controller**

**State: IDLE**

The controller is in the IDLE state until the trigger condition is met.

No outputs are asserted; load\_data and shift\_en are deasserted.

Transition: Trigger Received

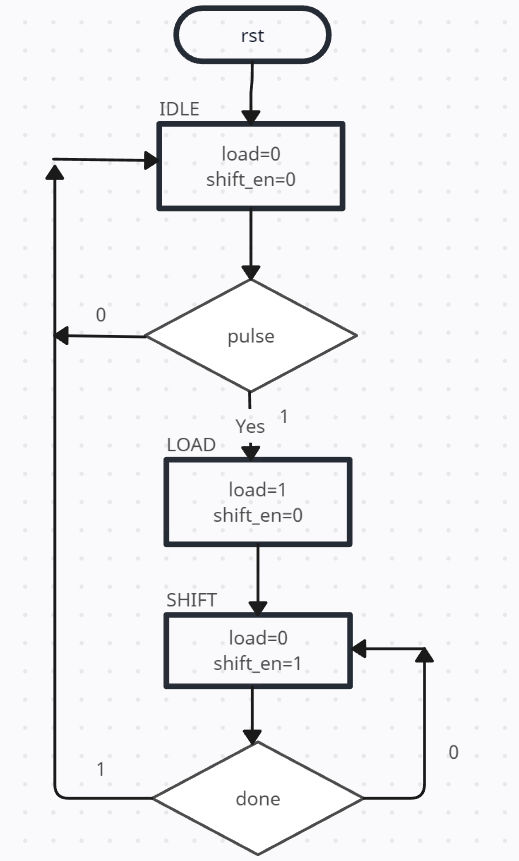
A decision point checks if the trigger to start the transmission is received. If Yes, the controller moves to the LOAD state.

If No, it remains in the IDLE state.

**State: LOAD**

In the LOAD state, load\_data is asserted to load the data into the shift register.

Once data is loaded, the controller moves to the SHIFT state.

****

**State: SHIFT**

In the SHIFT state, shift\_en is asserted to enable the shift register to shift out data.

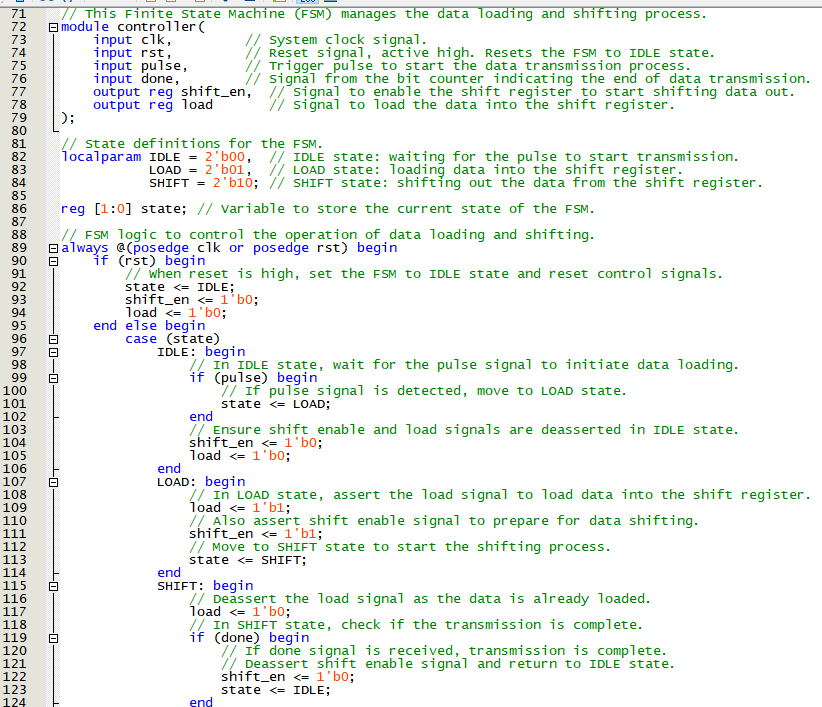
The controller remains in this state until the transmission is complete.

Transition: Transmission Complete A decision point checks if the done signal from the bit counter is asserted.

If Yes, the controller returns to the IDLE state, ready for the next transmission.

If No, it remains in the SHIFT state, continuing the transmission.

**Figure 7.ASM chart of controller**



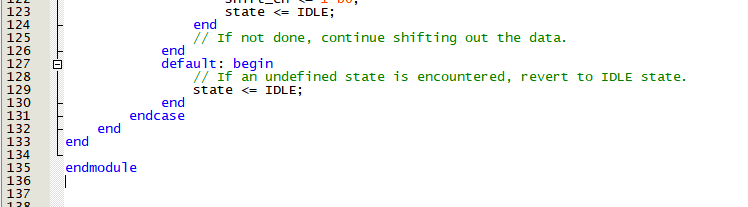


Figure 8 .verilog code with comment of controller

The central unit that orchestrates the overall operation. It manages the initiation and control of data transmission, coordinating the data flow and timing among the modules.

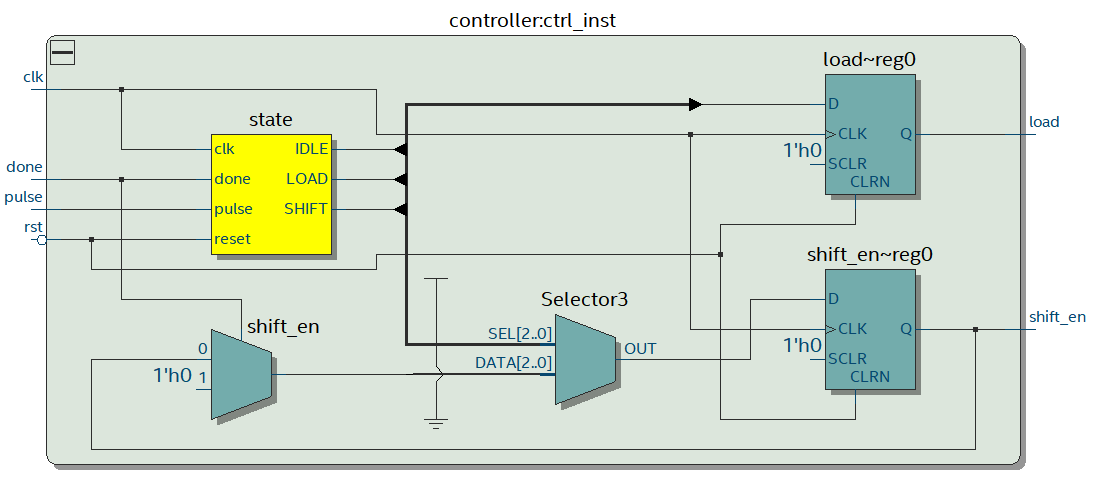


Figure 9 .RTL view of controller

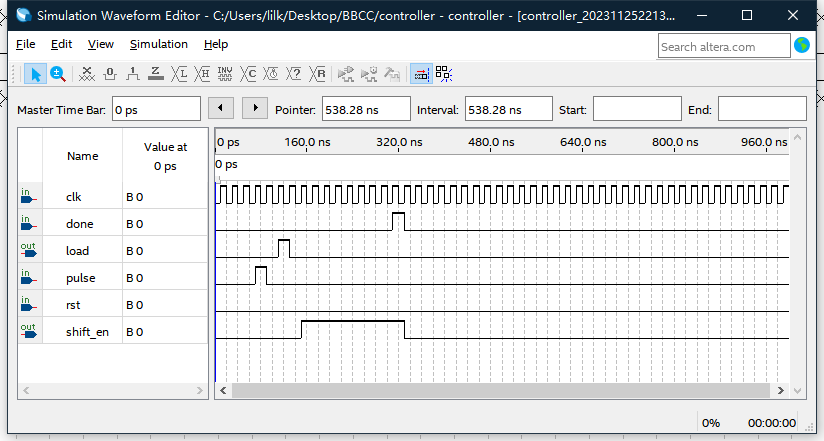


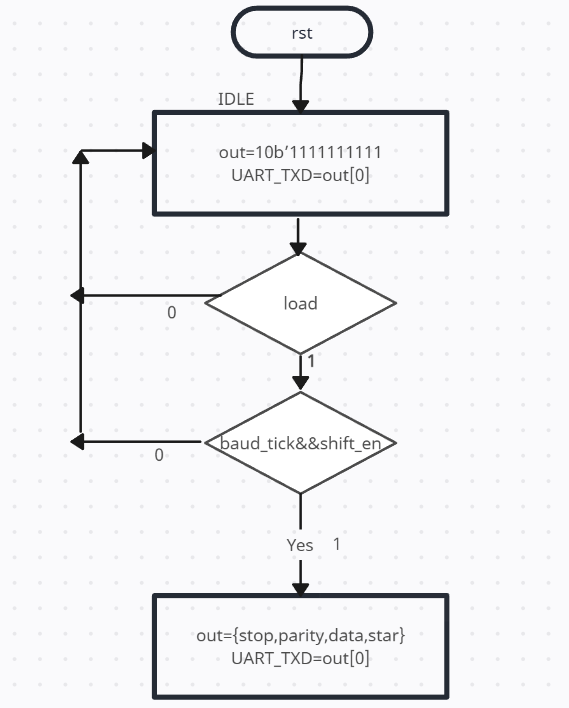
Figure 10 .Waveform Simulation of controller

In the controller module of the UART system, the process begins when a pulse signal is transmitted to the controller. This signal triggers a brief load pulse, which is then sent to the shift register. Upon receiving this pulse, the shift register starts loading the data, including the start bit, parity bit, data bits, and stop bit, preparing them for transmission.

After a clock cycle, the shift\_en signal is asserted and sent to both the shift register and the bit counter, activating them to begin their operations. The shift register starts shifting out the bits serially, beginning with the start bit, followed by the data and parity bits, and ending with the stop bit. Concurrently, the bit counter starts counting the bits as they are transmitted.

Once the transmission is complete, and the bit counter has counted all the bits of the frame, it generates a done signal which is fed back to the controller. Upon receiving the done signal, the controller deasserts the shift\_en signal, which stops the shifting process in the shift register and halts the operation of the bit counter. This action effectively terminates the current transmission cycle, resetting the system back to its initial state, ready to start a new transmission when triggered again.

**3.2 Shift Register**

****

**Figure 10.ASM chart of Shift Register**

**State: IDLE**

The shift register remains in the IDLE state until the load signal is received.

No output is transmitted **(txd** is idle).

Transition: Loading Data Upon receiving the load signal, the shift register transitions to the LOAD state.

**Signal: load**

In the load signal, the shift register loads the incoming data, setting up the start bit, data bits, parity bit, and stop bit. The register then transitions to the SHIFT state, ready to begin data transmission.

**Signal shift\_en and baud\_tick**

In the SHIFT state, data is shifted out on each baud tick if shift enable is active. The output bit **(txd)** is updated with each shift.

Transition: Completed Shifting

Once all bits have been transmitted, the register enters the IDLE state to reset and prepare for the next data load.

**Signal rst**

The shift register clears its contents and signals the completion of the transmission cycle. It then transitions back to the IDLE state.

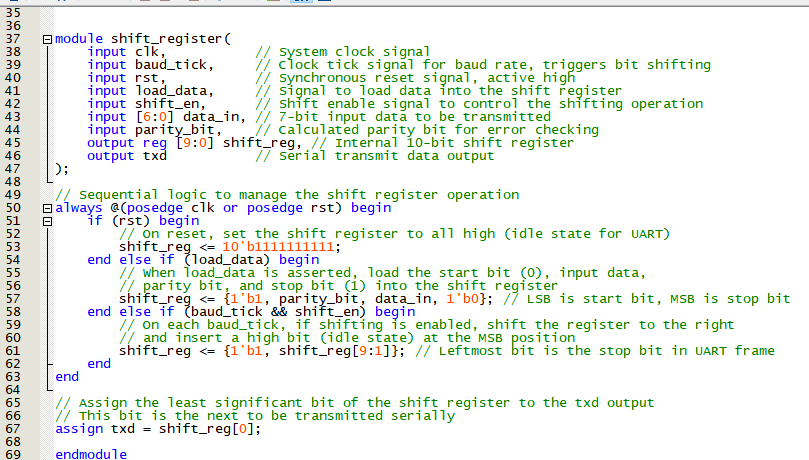


Figure 11.verilog code of Shift Register

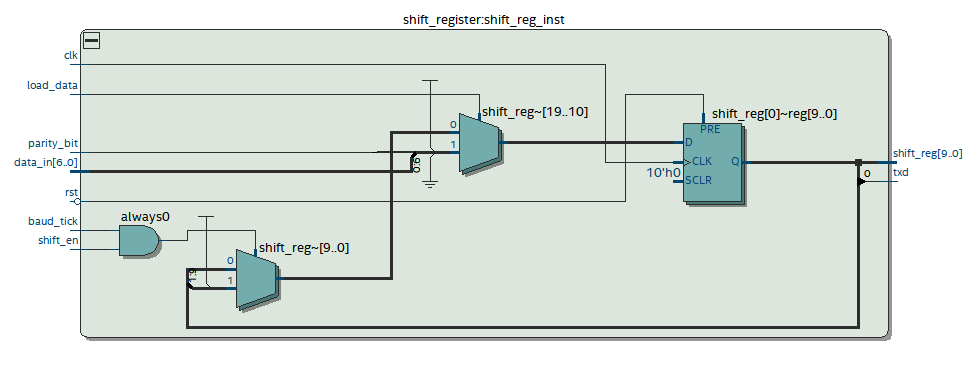
Responsible for storing and serially shifting out the data bits, including the start bit, data bits, parity bit, and stop bit, following the UART protocol. 

Figure 12.RTL view of Shift Register

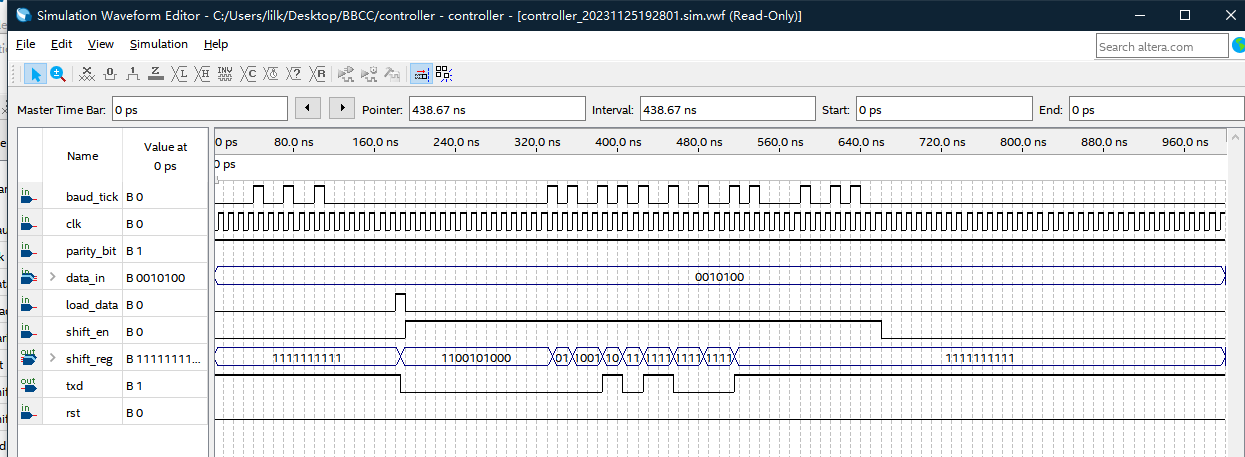
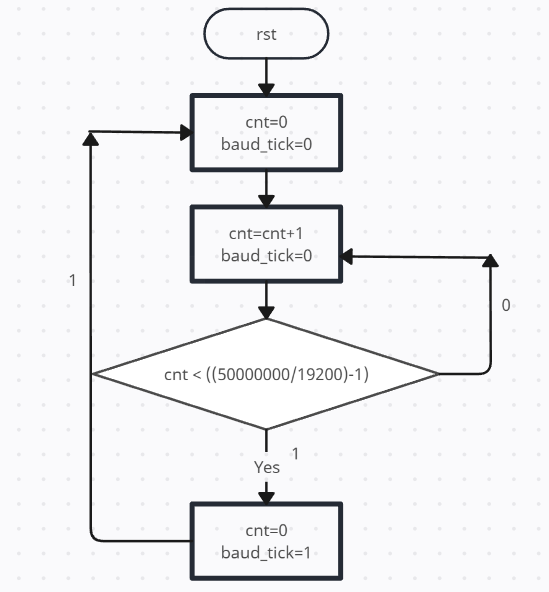


Figure 13 .Waveform Simulation of shift register

In this one simulation result, When the load signal is transmitted, The shift register starts loading the data, At this time, the simulated input data is '0010100', The data loaded in is '1100101000', The shiften signal is a high level, And when the baud tick signal comes, 1110010100, '1110010100', '1110010100', '1111001010', '1111100101', '1111110010 "1111111001" 1111111100 "1111111110" 1111111111'. The corresponding uart \_ txd output will correspond to '0', '0', '0', '0', '1', '0', '1', '0', '0', '1'

**3.3 Baud Rate Generator**



**Figure 14.ASM chart of baud\_rate\_generator**

**state: IDLE**

Initially, the baud tick generator is in the IDLE state, waiting for a reset or to start counting.

The baud tick output is low.

Transition: Start Counting

Upon reset deassertion, the system transitions to the COUNT state.

**Council signal: cnt**

In the cnt, an internal counter increments on each clock cycle .The state continues until the counter reaches a specified threshold, which is determined based on the desired baud rate.

Transition: Threshold Reached Once the counter hits the threshold, the system transitions to the baud\_tick

**signal: baud\_tick**

In the baud\_tick, the baud tick signal is asserted high for one clock cycle, indicating the time for a bit transmission in UART. After generating the tick, the system resets the counter and returns to the COUNT state to repeat the process.

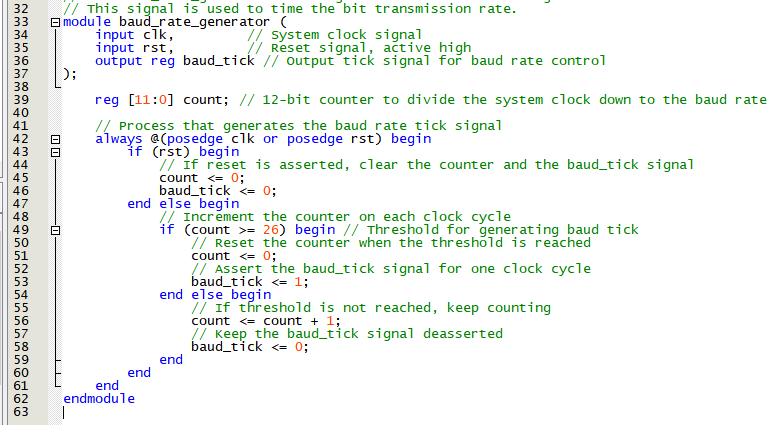


Figure 15..ASM chart of baud\_rate\_generator

Generates the timing signals (baud rate) to control the speed of data transmission, ensuring consistent and pre-set timing for transmitting data bits.

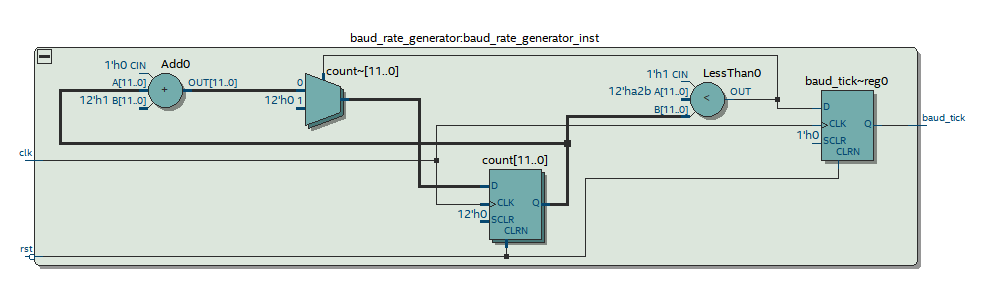


Figure 16.RTL view of baud\_rate\_generator

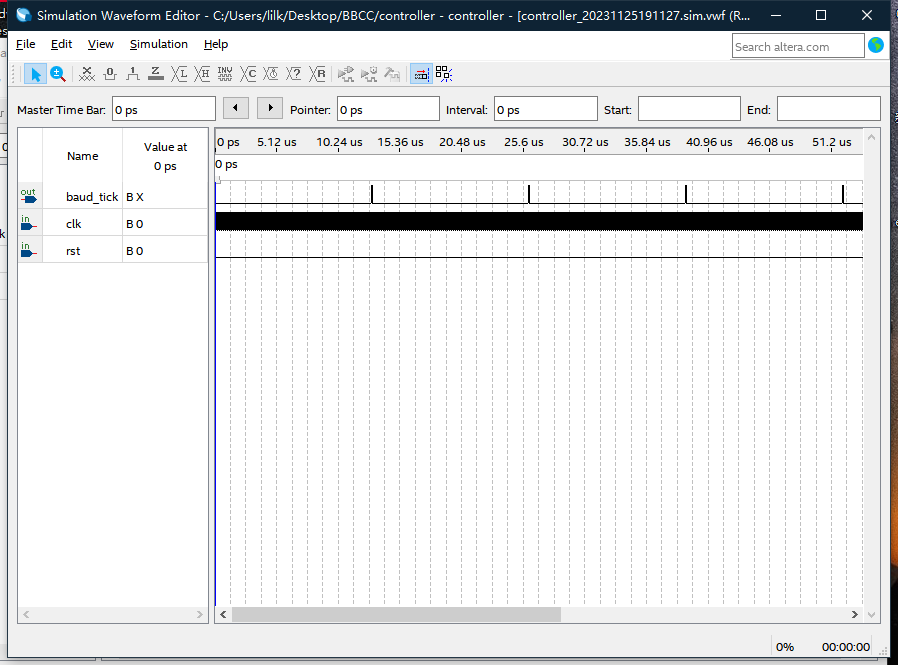


Figure 17 .Waveform Simulation of baud\_rate\_generator

Baud tick Is a single clock pulse generated with a port rate periodic clock of 1 / baud rate (s), Baud tick will be passed to the shift register and bit counter.

**3.4 Bit Counter**

**Initial State (IDLE):**

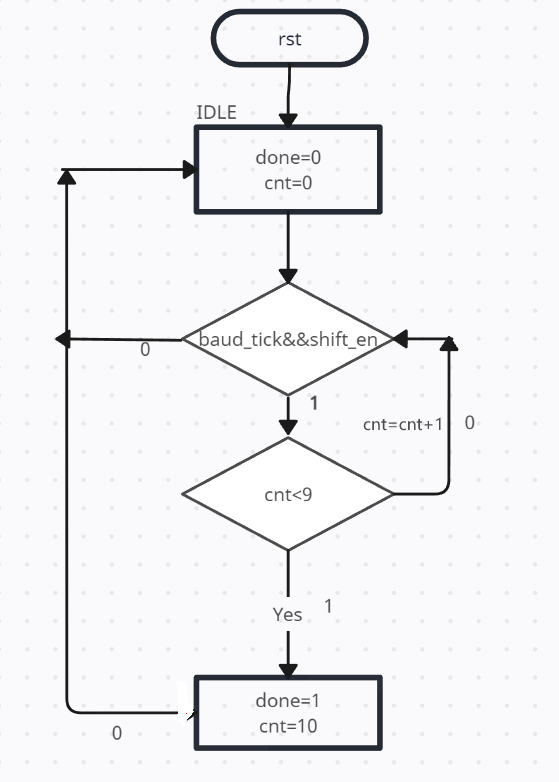
The chart starts with the system in the "IDLE" state. In this state, the counter (**cnt)** is set to 0, and the done signal is also set to 0, indicating that the system is ready to start a new transmission cycle. The system remains in this state until the reset (**rst)** condition is no longer active.

**Transition to Counting:**

From the "IDLE" state, when the reset **(rst)** is not active, and upon the conjunction of the baud\_tick and shift\_en signals being true (logical AND condition), the system transitions to the counting process. This transition is represented by the horizontal arrow pointing to the right, coming out of the decision diamond that checks the condition baud\_tick && shift\_en.

**Counting Process:**

During the counting state, with each baud\_tick while shift\_en is active, the counter (cnt) increments by 1. This represents the transmission of each bit in the UART frame.

****

**Figure 18.ASM chart of bit\_counter**

**Check Count:**

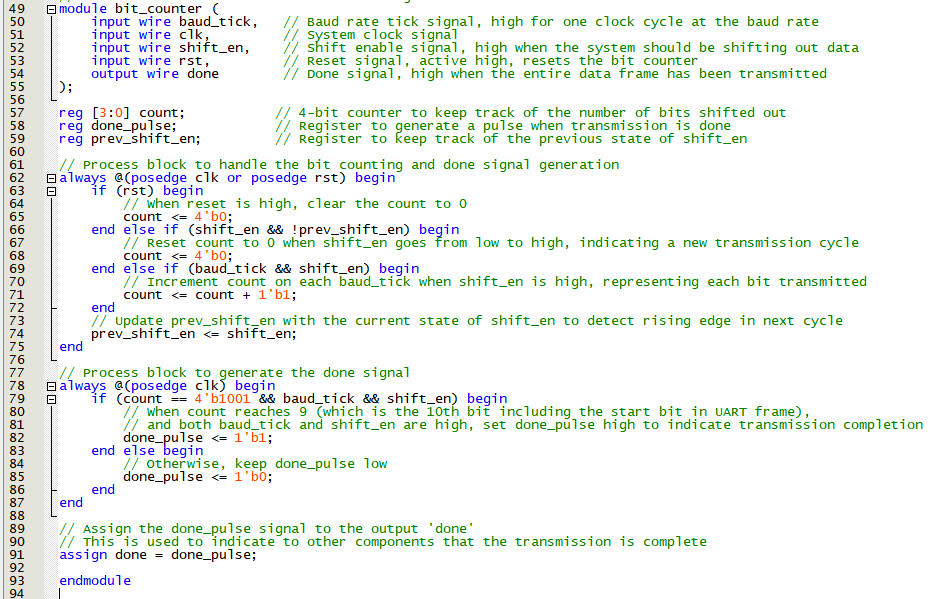
The system then checks if the counter has reached a count of **9**, which corresponds to the transmission of a complete UART frame (assuming one start bit, seven data bits, one parity bit, and one stop bit, for a total of ten bits). This condition is checked by the decision diamond labeled with **cnt<9.**

**Completion of Transmission:**

If the count is less than 9, the system loops back, incrementing the counter on each **baud\_tick**. This is indicated by the looped arrow pointing back to the counting process. Once the counter reaches 9, the system transitions to set the done signal to 1, indicating that the transmission of the current frame is complete. Simultaneously, the counter **(cnt)** is set to 10, likely as a part of the process to reset or prepare for the next frame.

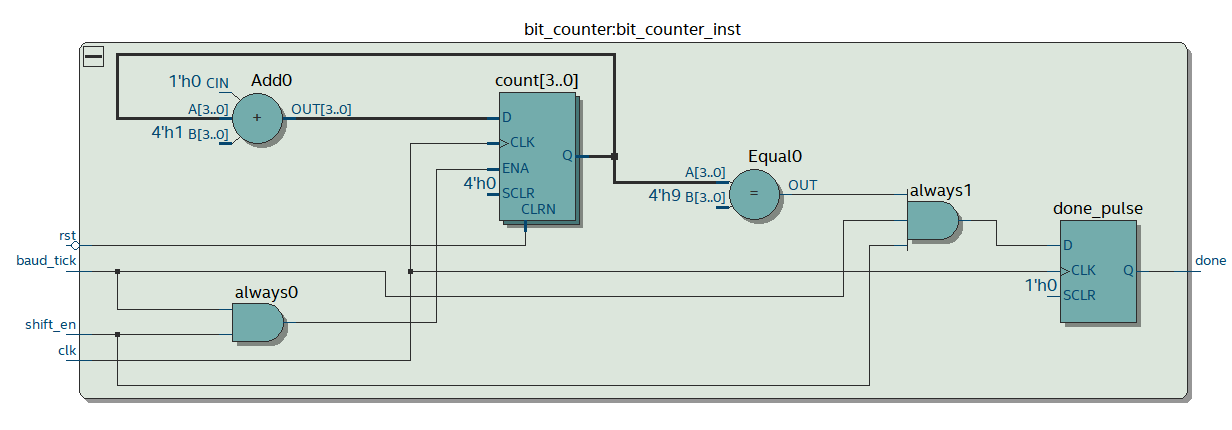
**Return to IDLE:**

After setting the done signal, the system returns to the "IDLE" state when the next reset (rst) occurs, as indicated by the arrow going up to the "IDLE" state, ready to start the next counting cycle for a new frame of transmission

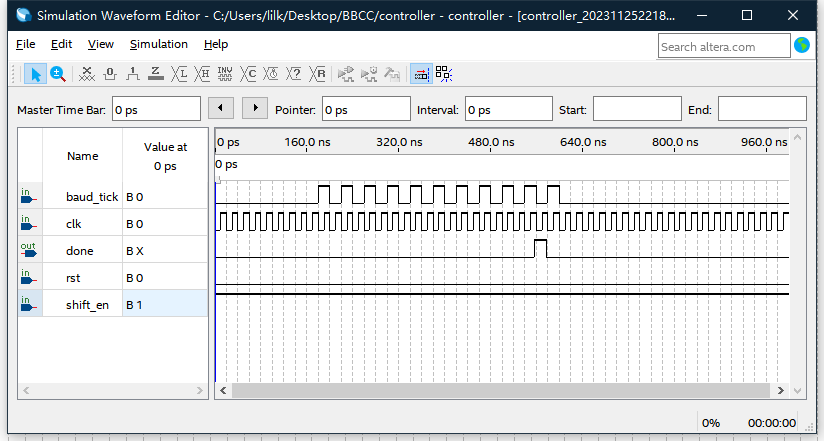


**Figure 19.Verilog code of bit\_counter**

Tracks the number of bits transmitted. It signals the completion of a UART frame transmission after the predetermined number of bits (including start, data, parity, and stop bits) is reached.



**Figure 20.RTL view of bit\_counter**



**Figure 21.Simulation of bit\_counter**

When the shift\_en signal comes, the bit counter is triggered and begins to count the coming baud tick pulses. When the number of pulses reaches 10, the don signal is triggered and transmitted to the controller.

**3.5 Even Parity Generator**

Since there is no decision-making or state transition, the equation is effectively just a representation of the operation.

**Operation:**

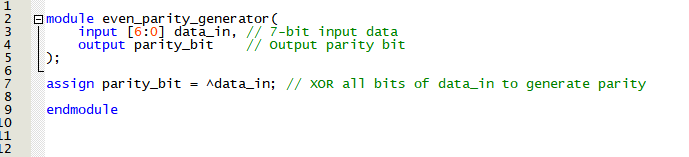
* **PARITY**: The only operation in this machine where the parity bit is computed.
* **Input: data\_in[6:0]:** The 7-bit input data for which the parity bit is to be generated.
* **Output:parity\_bit:** The output even parity bit.

**Operation Description:**

Upon receiving the 7-bit input, the even parity bit is computed by performing a bitwise XOR operation on all the bits of data\_in.

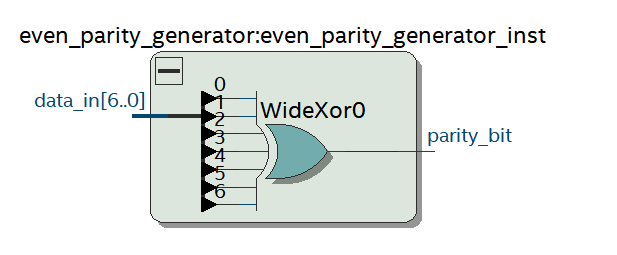
In an actual Verilog implementation, the even parity generator would be a one-liner code, as previously provided, because it's just a simple bitwise XOR operation, which is inherently a combinational logic operation.

**parity\_bit = data6​ ⊕ data5​⊕ data4​⊕ data3​⊕ data2​⊕ data1​⊕ data0​**



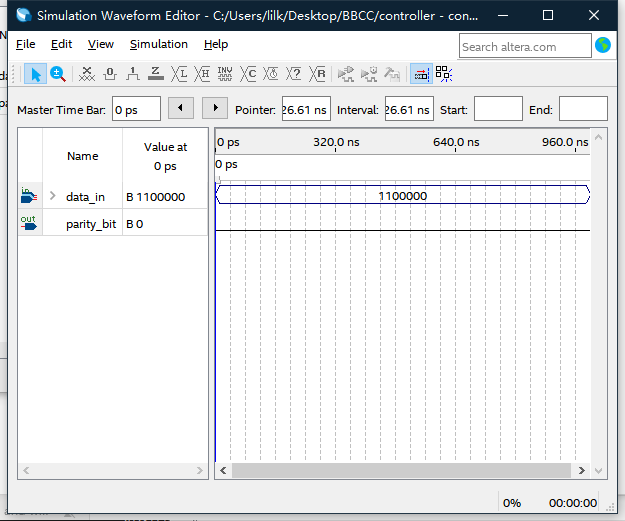
**Figure 22. Verilog code of bit\_counter**

Calculates the parity bit for error checking, ensuring an even number of '1' bits in the frame, including the parity bit.



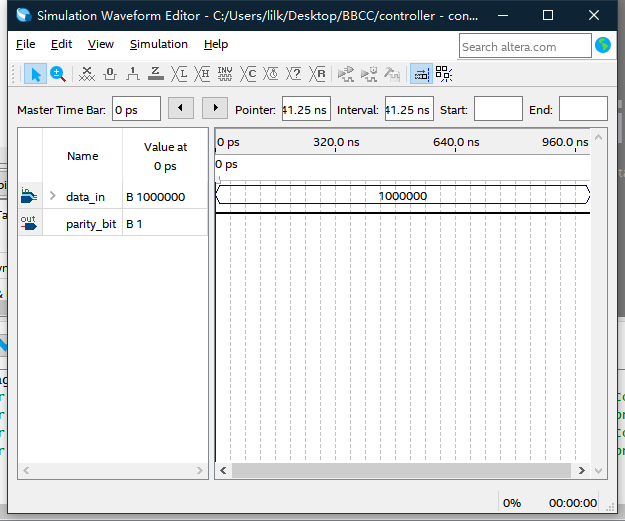
**Figure 23. Verilog code of bit\_counter**

This result shows the even parity will be ‘0’ if input is ‘1100000’,even number of ‘1’



**Figure 24.Simulation of bit\_counter**

This result shows the even parity will be ‘1’ if input is ‘1000000’, odd number of ‘1’



**Figure 24.Simulation of bit\_counter**

**3.6 Seven Segment Decoder**

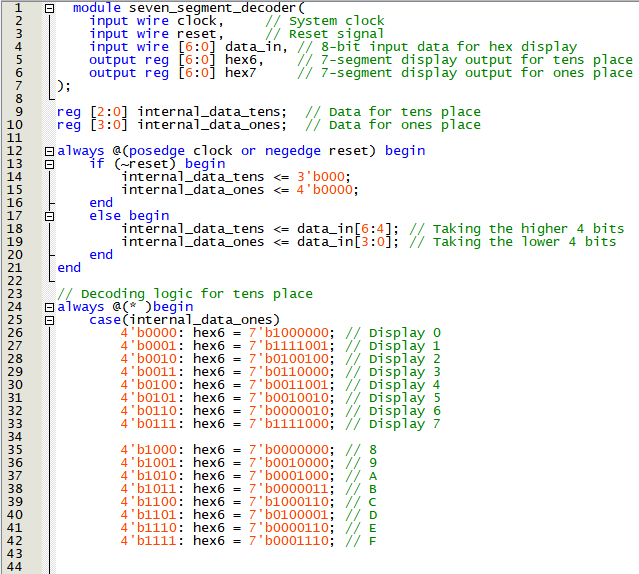
Converts the digital data into a format for a seven-segment display, providing a visual representation of the transmitted data, useful for monitoring and debugging.

**Operation Description**:Upon receiving the 4-bit input, the seven-segment display code is generated by mapping each possible input combination to the corresponding configuration of segments that represent the decimal digits 0 through 9 on the display.

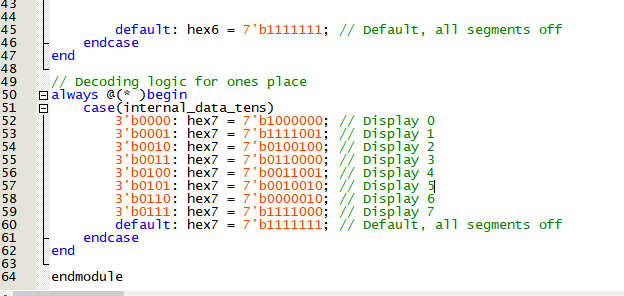
**DECODE:** The operation where binary input values are converted into seven-segment display codes.

**Inputs:data\_in[3:0]:** The 4-bit binary input that represents the number to be displayed.

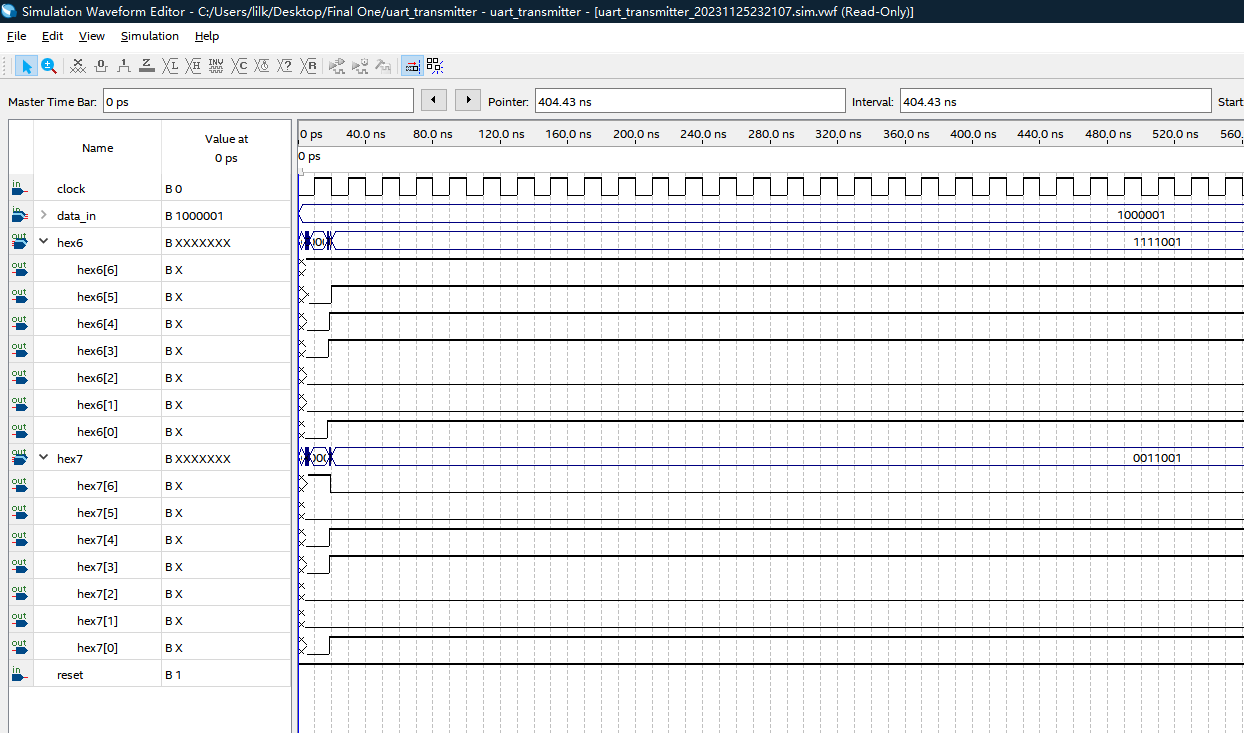
**Outputs:segments[6:0]**: The seven outputs corresponding to each segment in the seven-segment display, which will be turned on or off to represent the input number.



**Figure 25.verilog of Seven Segment Decoder**



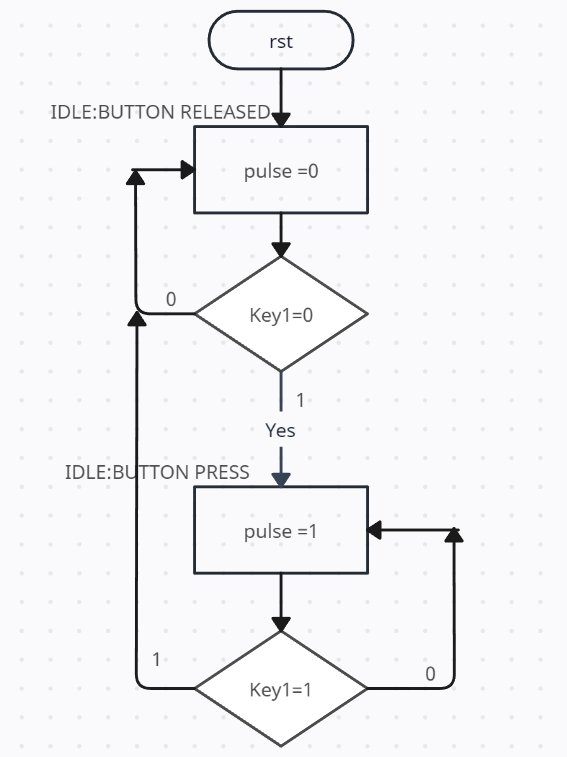
**Figure 26 verilog code ofSeven Segment Decoder**



**Figure 27.Simulation of Seven Segment Decoder**

For example, the input data bit 100 0001, the corresponding 16 base is 41, the result surface hex 7 and hex 6 will output '0011001' and '1111001', this is because the 7 digital tubes are cathode tubes, and the hex 7 and hex 6 on the development board will show the shapes of '4' and '1' to the operator.

**3.7 Single Pulse**



**Initial State (IDLE:BUTTON RELEASED):**

The system starts in the "IDLE:BUTTON RELEASED" state where pulse is set to 0. This state represents that the system is not currently detecting a button press, and the output pulse signal is inactive.

**Button Press Detection:**

From the initial state, the system checks the condition of Key1. If Key1 is 0 (representing a button press due to active-low logic), the system transitions to the "IDLE:BUTTON PRESS" state.

**Button Press State (IDLE:BUTTON PRESS):**

In the "IDLE:BUTTON PRESS" state, pulse is set to 1, generating a single output pulse. The system remains in this state as long as Key1 is 0.

**Button Release Detection:**

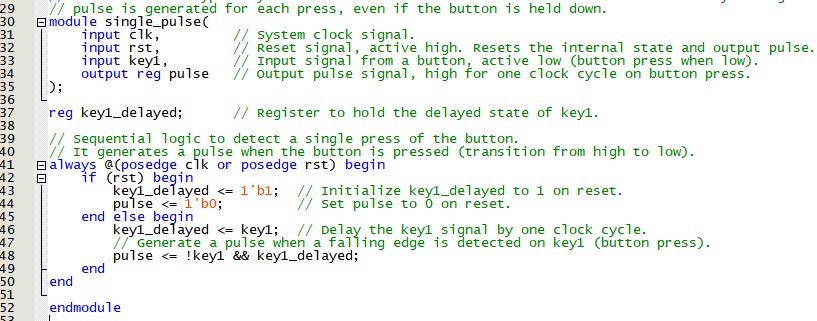
The system then checks if Key1 has returned to 1 (button release). If Key1 is 1, the system transitions back to the initial state "IDLE:BUTTON RELEASED", resetting pulse to 0 and waiting for the next button press event.

**Reset Condition**:

Independently of the button state, if a reset (rst) condition occurs, the system immediately transitions back to the "IDLE:BUTTON RELEASED" state.

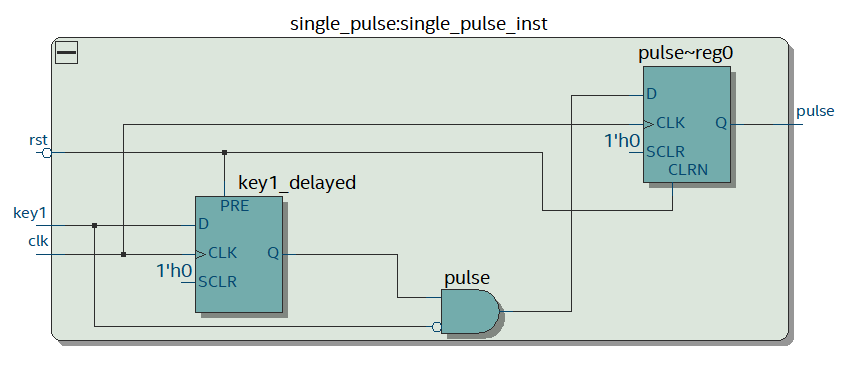
**Figure 28.ASM chart of Single Pulse**

**Single Pulse**

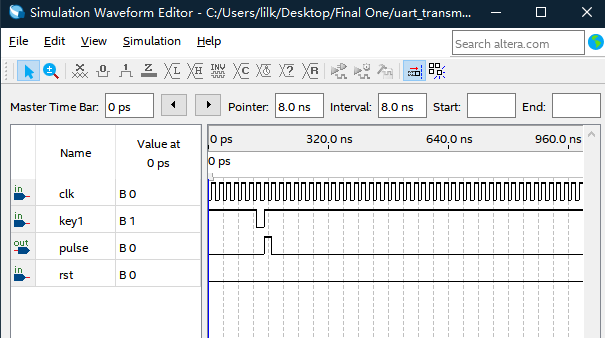


**Figure 29 verilog code of Single Pulse**

This module generates a single pulse signal, typically used to trigger the start of data transmission. It acts upon a specific user action or condition, such as pressing a button, to initiate the transmission process.



**Figure 30.RTL view of Single Pulse**



**Figure 31.Simulation of Single Pulse**

**clk:** The clock signal that provides the timing reference for the circuit.

**key1**: An input signal that likely represents a button press; an active low signal where '0' indicates the button is pressed and '1' indicates it is not pressed.

**pulse**: The output signal which should generate a single pulse when the button (key1) is pressed.

**rst**: A reset signal that, when active (typically high), would reset the circuit to an initial state.

Based on the waveform patterns in the screenshot and the typical operation of a single pulse circuit, here's a description of the simulation results for the single pulse functionality:

Initially, all signals (key1, pulse, and rst) are at '0', indicating an idle state with no button press, no pulse output, and no reset condition.

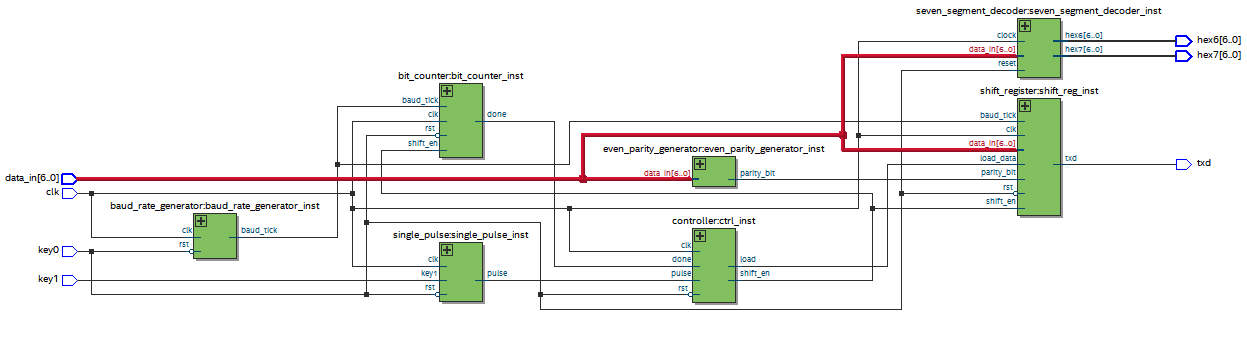
At some point in the simulation, key1 transitions from '1' to '0', which suggests a button press occurs. This transition should ideally trigger the single pulse circuit.

Following the button press, the pulse signal should exhibit a single transition from '0' to '1' and then back to '0', representing the single pulse generated due to the button press.

The rst signal remains at '0' throughout the visible part of the simulation, indicating that no reset has been triggered and the circuit continues its operation.

**4.Schematic of the full system**

* **Controller Module:** This is the central unit that orchestrates the UART transmission process, deciding when to load data into the shift register and when to start and stop transmitting bits.
* **Shift Register Module**: This module holds the data to be transmitted and shifts it out serially, bit by bit, under the control of the controller.
* **Baud Rate Generator Module:** This generates the clock signals (baud rate ticks) that determine the timing of bit transmissions.
* **Bit Counter Module**: It counts the number of bits that have been transmitted to ensure that the entire data frame, including start, data, parity, and stop bits, is sent out.
* **Even Parity Generator Module**: If parity is used, this module calculates an even parity bit for the data frame.
* **Single Pulse Generator Module:** Often used to clean up the signal from a button press to start the transmission.
* **Seven Segment Decoder Modul**e: If the system includes a display component, this module converts binary inputs into signals that can drive a seven-segment display for user feedback.

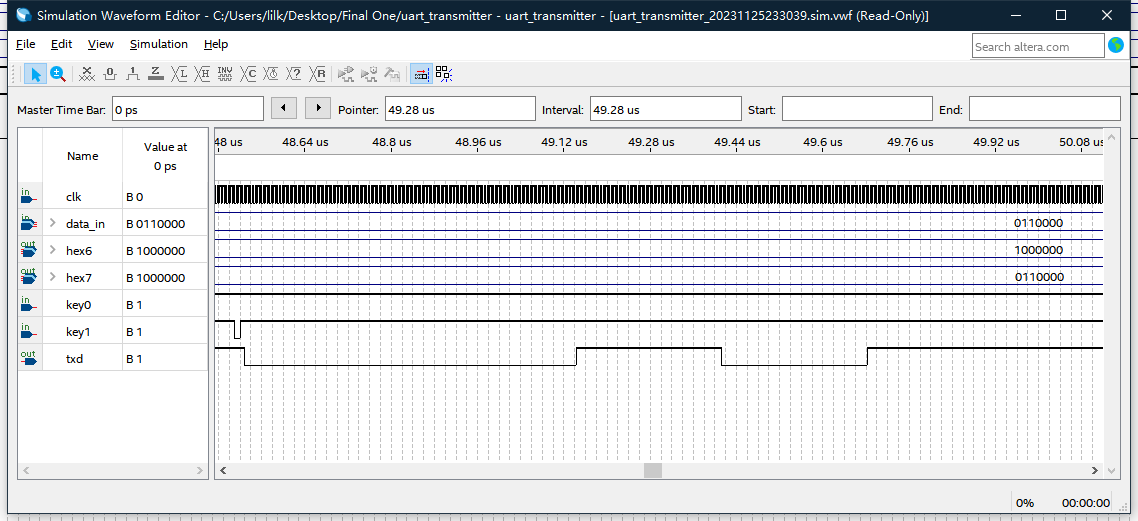


**Figure 32. Full system RTL view**

**In the schematic:**

* The Controller Module would have connections to the Shift Register Module, the Bit Counter Module, and possibly the Single Pulse Generator Module.
* The Shift Register Module would receive the data to be transmitted as well as control signals from the Controller Module and the Baud Rate Generator Module.
* The Baud Rate Generator Module would be connected to the Controller Module and the Shift Register Module to provide the timing signals.
* The Bit Counter Module would be connected to the Shift Register Module to count the bits as they are transmitted and would signal the Controller Module when transmission is complete.
* The Even Parity Generator Module would be connected to the data input and the Shift Register Module to insert the parity bit into the data frame.
* The Seven Segment Decoder Module would be connected to a data source to display and possibly to the Controller Module to indicate the status of the system.

**5.Simulation of the full system.**



The **clk** signal is oscillating regularly, providing the timing reference for the system.

**data\_in** is set to **'01100000'**, indicating the data ready to be transmitted.

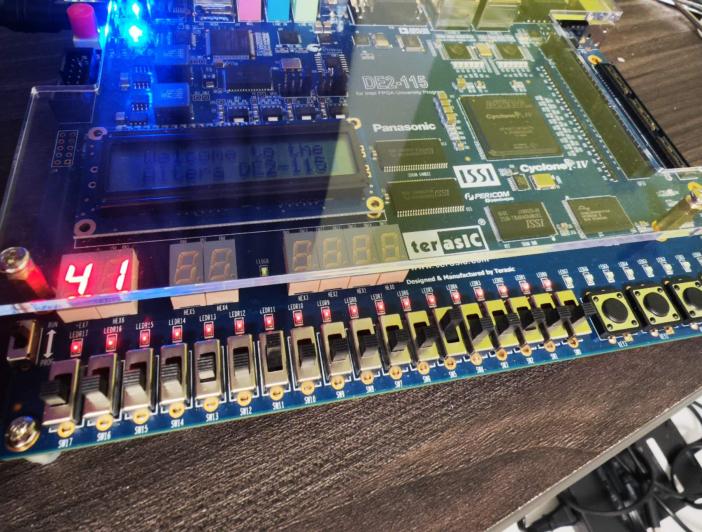
**hex6 and hex7** are constant at **'1000000' and ‘0110000’**, likely representing a value on a seven-segment display, not changing during this part of the simulation.

**key0** is high (**'1')**, indicating no reset event is occurring.

**key**1 transitions from high to low and back to high, suggesting a button press and release.

The **txd line**, which should show the serial data being transmitted, remains high **('1'),** indicating that either the transmission has not started yet or this part of the waveform does not include the actual data transmission

**6.Explanation of experimental test results.**



**Displaying '41' on a Seven-Segment Display:**

* SW 11 and SW 5: These are likely switches on an FPGA board. Dialing SW 11 and SW 5 to specific positions sets the binary input that corresponds to the hexadecimal number '41'.
* Seven-Segment Display: This display converts the binary input from the switches into a human-readable form. The binary-to-hexadecimal conversion is visualized by illuminating specific segments on the display to show the number '41'.

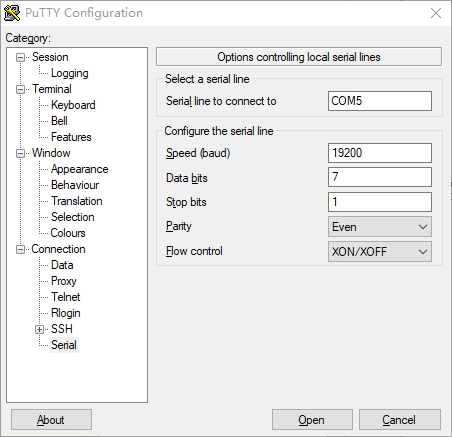
**Configuring the Serial Port:**

* Baud Rate: The communication speed of the serial port is set to 19200 bits per second.
* Parity Check: An even parity check is configured, meaning the parity bit will be set so that the total number of 1s in the data packet is even.
* Data Bits: The data packet is configured to have 7 data bits, which is common for ASCII character transmission.

**Sending Data via UART:**

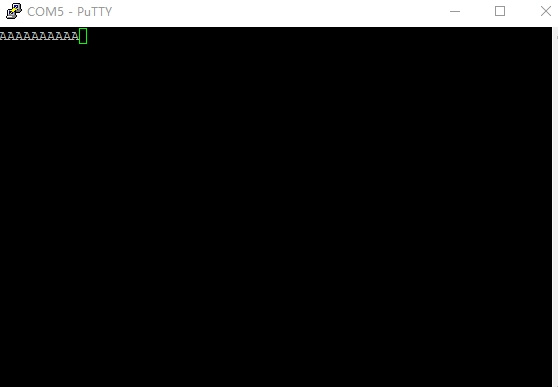
* Key1: This is a button on the FPGA board used to initiate the transmission of data.
* Transmitting 'A': When Key1 is pressed, the character 'A' is sent over the serial port. Since the ASCII value for 'A' is 65 (0x41 in hexadecimal), this correlates with the '41' displayed on the seven-segment display.
* Serial Port Display: A terminal program (like PuTTY or Tera Term) connected to the serial port will display the character 'A', confirming that the data has been transmitted correctly.
* By dialing sw 11 and sw 5, the seven-segment digital tube will display 41. This step is to convert the binary input into a hexadecheal visual effect

**Figure 33. HEX value of input data**



* The configuration of the serial port, for example, the port rate is set to 19200, the check mode is an even check, and the received input is a 7-bit data bit

**Figure 34. configuration of putty**



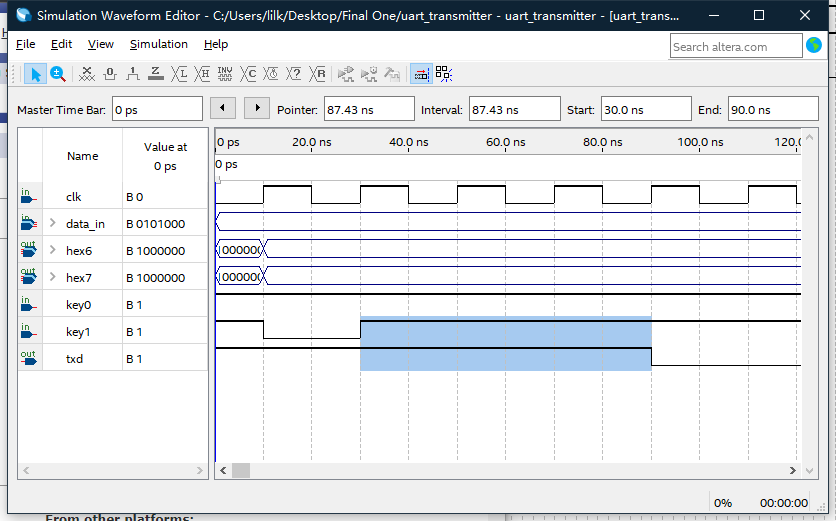
**Figure 35. ASCII display on putty**

By pressing key1 to send the data, the letter 'A' will be displayed on the serial port, and the results can be proved to be expected by comparing the ASCII symbol

1. **Discussion/Conclusions**

The design's primary goal was to demonstrate the functioning of a UART transmitter system and its interaction with a user interface, such as a seven-segment display and input switches. The specific output of displaying the letter 'A' on a terminal like PuTTY served as a sample output to validate the entire communication process from input to display.

The system's capability to convert binary switch inputs to a hexadecimal display on a seven-segment display is a clear indication of the successful integration of digital logic with user interfaces. This conversion process is crucial for providing visual feedback to the user and confirming that the system accurately interprets binary inputs. The UART configuration's successful implementation—baud rate, parity check, data bits—is confirmed by the correct transmission and reception of data.



In conclusion, the experiment's outcomes exceed the mere display of a character; they encompass the successful demonstration of a UART transmission system that has been meticulously designed, implemented, and tested to meet industry-standard communication protocols, However, when comparing the time interval between the input and the output, the system will find a delay of 3 clk. For this problem, students can eliminate the delays by optimizing the design.