## UART Transmitter Module

This module is all about getting a basic **UART transmitter** up and running. It uses the **8N1 format**, which stands for:

* **8 data bits**
* **No parity bit**
* **1 stop bit**

It sends serial data at **9600 baud**, with the timing derived from a **12 MHz oscillator**. The module runs on a simple state machine and includes some bonus visuals—like blinking RGB LEDs—to make sure you can tell it’s doing something!

**1. Understanding the Code**

**Top Module: The Brain of the Operation**

The **top module** pulls together several components:

* A 12 MHz internal clock
* A clock divider to generate a 9600 Hz signal
* The UART transmission logic
* A small RGB LED control system

Here’s how each part plays its role:

* **Clock Generation:** It starts with an internal **12 MHz oscillator**.
* **Clock Division:** That frequency is divided by **1250**. Since UART clocks need to toggle every half-period, the actual toggle happens every **625 cycles**. This results in a clean **9600 Hz clock**, which matches our baud rate.
* **UART Transmission:** The transmitter keeps sending the character 'D' over and over using the **8N1** format.
* **LED Blinking:** The RGB LEDs are connected to a counter. Different bits of this counter toggle the LEDs, so they blink at regular intervals, giving you a visual confirmation that the system is active.

In short, this module handles timing, transmits a test character, and blinks LEDs—all in sync.

**uart\_tx\_8n1 Module: How Data is Sent**

The actual transmission logic lives in the **uart\_tx\_8n1** module. It uses a **Finite State Machine (FSM)** to go through the steps of sending each bit of the UART frame.

**Baud Rate Generator: Getting to 9600 Baud**

Here’s how it creates the correct timing signal:

* A counter increases with every tick of the **12 MHz clock**.
* When it hits **1249**, it resets and toggles a signal called baudclk\_en.
* Since this toggle happens every 1250 cycles, it results in a **9600 Hz signal**, which matches the required UART baud rate.

**State Machine Breakdown: Step-by-Step Transmission**

The FSM moves through different states to send a full UART frame:

**1. IDLE (STATE\_IDLE)**

* When senddata = 1:
  + The module moves to STATE\_STARTTX
  + The byte to be sent (txbyte) is loaded into a buffer called buf\_tx
  + txdone is cleared, meaning transmission is now in progress
* If senddata = 0, it stays idle:
  + txbit stays high (UART line idles high)
  + txdone stays low

**2. START BIT (STATE\_STARTTX)**

* txbit is set to **0** to indicate the start of transmission
* The system then moves to STATE\_TXING

**3. SENDING DATA (STATE\_TXING)**

* As long as bits\_sent < 8:
  + The least significant bit (LSB) of buf\_tx is sent out through txbit
  + buf\_tx is shifted right to prepare the next bit
  + bits\_sent is increased by 1

**4. STOP BIT (STATE\_TXDONE)**

* After all 8 data bits are sent:
  + txbit is set to **1** (stop bit)
  + bits\_sent is reset
  + State changes to STATE\_TXDONE

**5. DONE → BACK TO IDLE**

* In STATE\_TXDONE, the system:
  + Sets txdone = 1 to indicate the frame is fully transmitted
  + Moves back to STATE\_IDLE, ready for the next byte

**2. System Architecture**

**A diagram of a machine

Description generated with high confidenceBlock Diagram**

This diagram shows how the modules are wired together: oscillator, clock divider, UART logic, and LED driver.

**A diagram of a computer

Description generated with high confidenceCircuit Diagram**

**A diagram of a computer

Description generated with high confidence**

This shows how the components are connected physically—great for when you're wiring it up on your FPGA board.

**3. Synthesis & Programming**

Once the code is ready, here’s how to test everything:

**Step 1: Clone the Repo**

bash

CopyEdit

git clone

**Step 2: Build the Bitstream**

bash

CopyEdit

make build

This command compiles the project and creates top.bin, which you’ll load onto your FPGA.

**Step 3: Flash It to the FPGA**

bash

CopyEdit

sudo make flash

This uploads the design onto your FPGA board.

**Step 4: Open the UART Terminal**

bash

CopyEdit

sudo make terminal

Once this runs, you should see the letter **'D'** appearing repeatedly. That’s your UART transmission in action—sending data at **9600 baud**, just as expected.