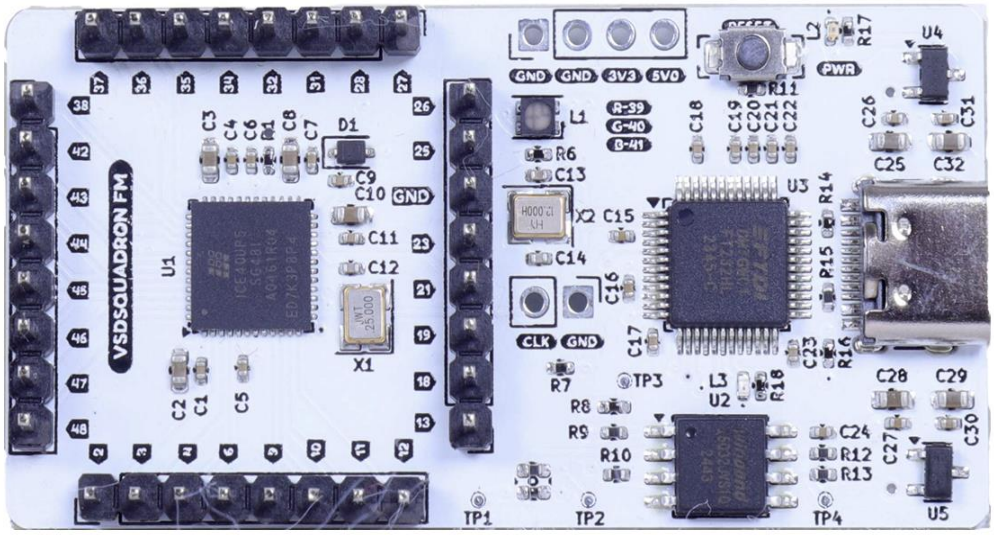


VSDFPGA Task 3

UART TRANSMIT

SUBMITTED BY

ISHA SINGH

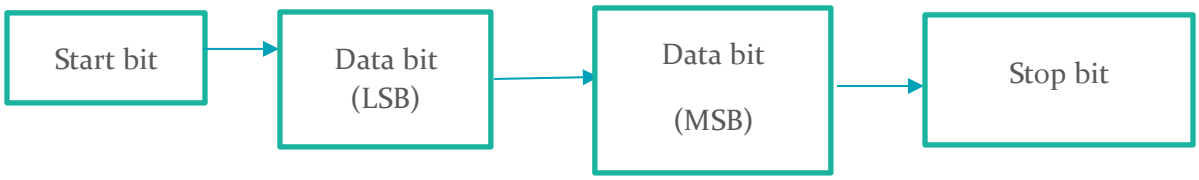
Introduction

The Serial communication is a fundamental requirement in embedded and FPGA-based systems for exchanging data with external devices such as personal computers, microcontrollers, and sensors. One of the most widely used serial communication protocols is UART (Universal Asynchronous Receiver Transmitter) due to its simplicity and minimal hardware requirements.

This project focuses on the design, implementation, and verification of a UART transmitter module using Verilog HDL on an FPGA. The UART transmitter follows the 8N1 format, which consists of 8 data bits, no parity bit, and 1 stop bit, and operates at a baud rate of 9600 bps. The system transmits a fixed ASCII character ("D") periodically to an external UART-compatible device. Additionally, RGB LEDs are used to provide visual feedback of internal activity

.

UART TRANSMIT



1. Idle State

In the idle state, the UART transmitter is not sending any data and the transmission line (TX) is held at a logic HIGH (1). This state ensures that the receiver can detect the beginning of a new transmission when the line transitions from HIGH to LOW. The transmitter continuously remains in this state until a transmission request is received.

2. Start Bit

When the senddata signal becomes high, the UART transmission begins by sending a **start bit**, which is a logic LOW (0). This falling edge informs the receiving device that a new data frame is about to be transmitted. The start bit synchronizes the receiver to sample incoming data at the correct time.

3. Data Bits

After the start bit, the transmitter sends **8 data bits**, starting from the **least significant bit (LSB)** and ending with the most significant bit (MSB). The data bits are shifted out one by one using a shift register (buf_tx), with each bit transmitted on a separate clock cycle of the baud-rate clock. This ensures proper timing and accurate serial communication.

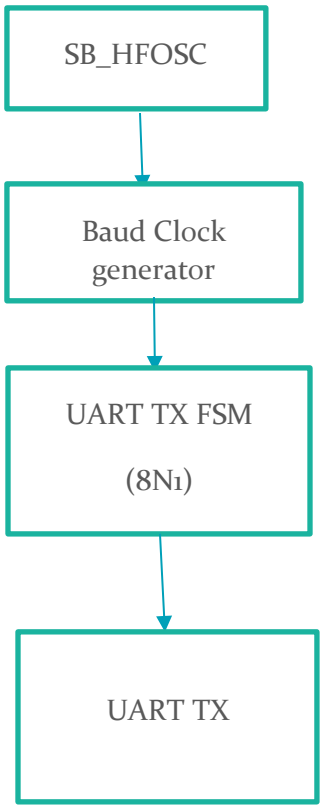
4. Stop Bit

Once all 8 data bits are transmitted, the transmitter sends a **stop bit**, which is a logic HIGH (1). The stop bit marks the end of the UART data frame and allows the receiver time to process the received byte. After the stop bit, the TX line remains HIGH, returning the system to the idle condition.

5. Done State

In the done state, the transmitter asserts the txdone signal to indicate that the entire data frame has been successfully transmitted. The internal state machine then resets its counters and transitions back to the idle state. This prepares the transmitter for the next transmission cycle.

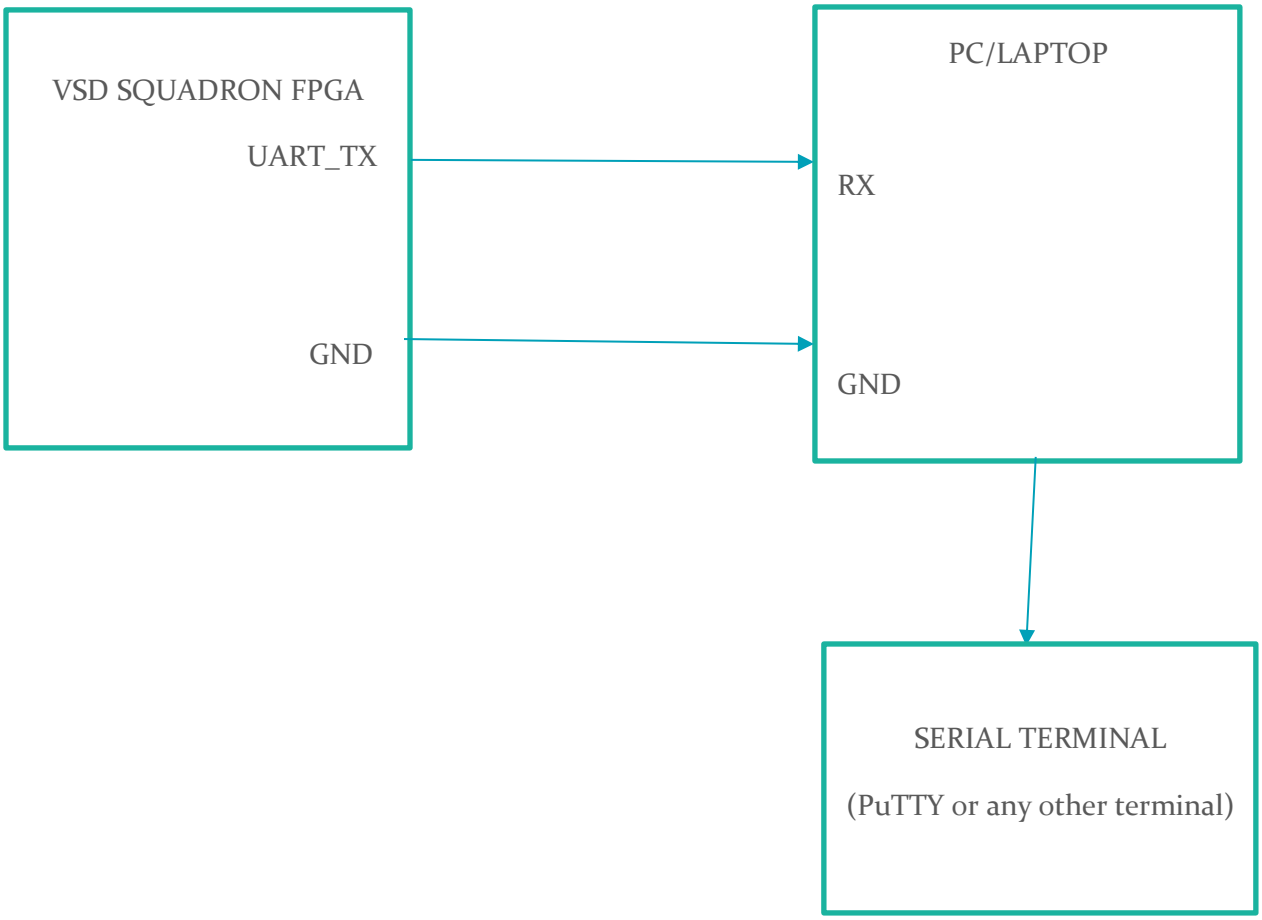
BLOCK DIAGRAM UART TRANSMIT



The UART transmitter system consists of an internal oscillator, a baud rate generator, a UART control unit, and an output transmission pin. The internal oscillator generates the main clock required for FPGA operation. This clock is divided by the baud rate generator to

produce a precise 9600 Hz clock for UART timing. The UART transmitter block uses this clock to control a finite state machine that sends the start bit, data bits, and stop bit in sequence. The serialized data is then transmitted through the UART TX pin to an external UART-compatible device such as a PC. This modular structure ensures reliable and standardized serial communication.

CIRCUIT DIAGRAM



The circuit illustrates the connection between the FPGA and a PC using a USB-to-UART adapter for serial communication. The UART transmit (TX) pin of the FPGA is directly connected to the receive (RX) pin of the USB-UART adapter, enabling one-way data transmission from the FPGA to the external device. A common ground connection between the FPGA and the adapter is essential to ensure a shared voltage reference and reliable signal transmission. The USB-UART adapter converts the TTL-level serial data from the FPGA into USB format. The PC detects this adapter as a virtual COM port. Serial terminal software

such as **PuTTY** is then used to open this COM port. The transmitted UART data is displayed on the serial terminal window. This setup allows easy testing, monitoring, and verification of the UART transmitter functionality using minimal hardware.

SUMMARY OF VERILOG CODE WALKTHROUGH

1. Module Declaration

The top module defines the interface between the FPGA and external devices. It has three output pins for RGB LEDs (led_red, led_green, led_blue) and one output for the UART transmission (uarttx). There is a single input, hw_clk, though the design primarily uses the FPGA's internal oscillator for clock generation. This structure allows the module to transmit UART data while simultaneously providing visual feedback using the LEDs.

2. Internal Oscillator

The code instantiates the FPGA's **high-frequency internal oscillator**, which serves as the main clock source for the system. By setting the division factor (CLKHF_DIV) to ob10, the oscillator output frequency is divided appropriately for subsequent clock division. This eliminates the need for an external crystal, simplifying the hardware setup.

3. Frequency Counter

reg [27:0] frequency_counter_i;

A 28-bit counter is implemented that increments with each rising edge of the internal oscillator. This counter serves multiple purposes: it triggers periodic UART transmissions and generates the patterns for the RGB LEDs. By monitoring specific bits of the counter, the design can control timing for both UART transmission and LED display without additional clocks.

4. Baud Rate Clock Generation (9600 Hz)

The code generates a **9600 Hz clock** necessary for UART communication. A counter (cntr_9600) increments on each oscillator pulse, and when it reaches period_9600, the clk_9600 toggles, creating a stable baud-rate clock. This clock is then used by the UART transmitter module to ensure accurate timing of start, data, and stop bits.

5. UART Transmitter Instantiation

The UART module uart_tx_8n1 is instantiated to transmit a fixed ASCII character "D". The transmission is triggered periodically using a specific bit of the frequency counter

(frequency_counter_i[24]). The transmitter outputs the serial data to the uarttx pin, which can be connected to a PC or any UART-compatible device.

6. RGB LED Driver

The RGB LED driver provides a visual indication of system activity. Different combinations of counter bits drive the green, blue, and red LEDs, creating dynamic patterns. This confirms that the FPGA is running correctly and the internal oscillator is active, providing both functional and diagnostic feedback.

7. UART Transmitter Module (uart_tx_8n1)

The UART transmitter module implements a **Finite State Machine (FSM)** with four states: IDLE, STARTTX, TXING, and TXDONE.

1. **IDLE:** The TX line remains HIGH, waiting for a transmission trigger.
2. **STARTTX:** A LOW signal is sent to indicate the start of a frame.
3. **TXING:** The 8-bit data is transmitted LSB first using a shift register.
4. **TXDONE:** The stop bit is sent, and the module signals transmission completion through txdone.

The FSM ensures proper sequencing and timing for each UART frame, making the transmission reliable.

8. Data Shifting and Bit Transmission

During the TXING state, the module shifts out each bit of the byte using buf_tx. Each clock pulse moves the next bit into position, ensuring the receiver samples data correctly. After all 8 bits are sent, a stop bit is transmitted, completing the frame.

Integration Steps and Observations While Working with the VSDSquadron FPGA Mini Board

1. The VSDSquadron FPGA Mini board was connected to the host computer using a USB Type-C cable, as specified in the datasheet. Verified that the USB cable supported **data transfer** (not power-only)
2. Ensured the board's power LED was ON
3. Confirmed proper physical seating of the USB-C connector
4. After connecting the board, the system was checked to ensure the FPGA Mini board was detected by the operating system.

5. lsusb command was used

6. The output confirmed the presence of an **FTDI device**, indicating that the onboard programmer was recognized successfully.

7. As specified in the VSDSquadron FPGA Mini datasheet, the open-source iCE40 FPGA toolchain was installed. This included:

- Yosys (for Verilog synthesis)
- nextpnr-ice40 (for place and route)
- icestorm utilities (icepack, iceprog)

8. The provided Makefile was used to automate the build and flash process.

make clean: This command removed previously generated files such as .json, .asc, and .bin, ensuring a clean build environment.

make build: This step performed the Verilog synthesis, place and route, bitstream generation.

9. Successful execution resulted in the generation of the FPGA programming file (.bin).

10. Once the build completed successfully, the bitstream was flashed onto the FPGA Mini board using:

sudo make flash

11. Root privileges were required to access the USB programming interface.

12. The flashing process wrote the bitstream into the SPI flash memory on the board, enabling non-volatile storage of the design.

13. Immediately after flashing, the FPGA was disconnected from the Virtual Machine and then the PuTTY terminal was opened. In the Device Manager, the Port was checked and then in PuTTY after selecting the serial connection and setting the baud rate to 9600, the letter "D" was flashed continuously on the PuTTY terminal.

14. This confirmed successful transmission.

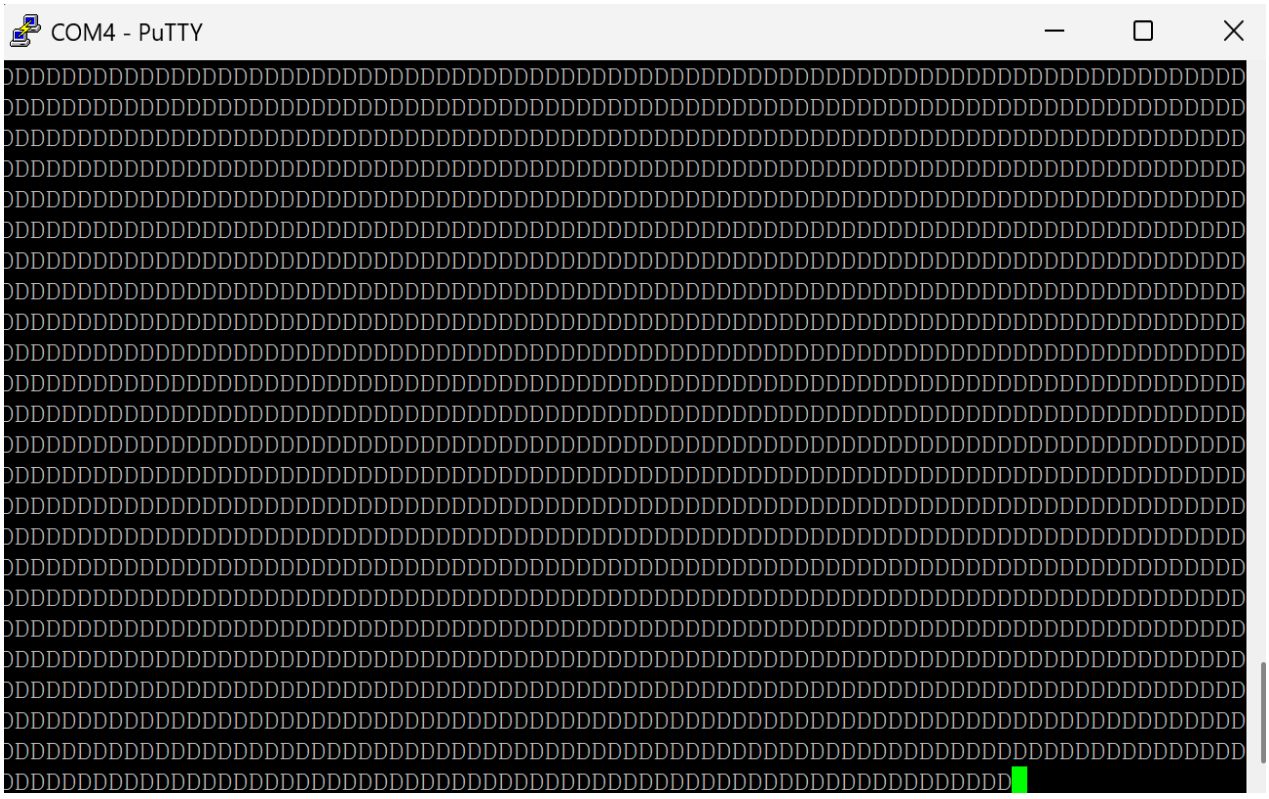
Observations:

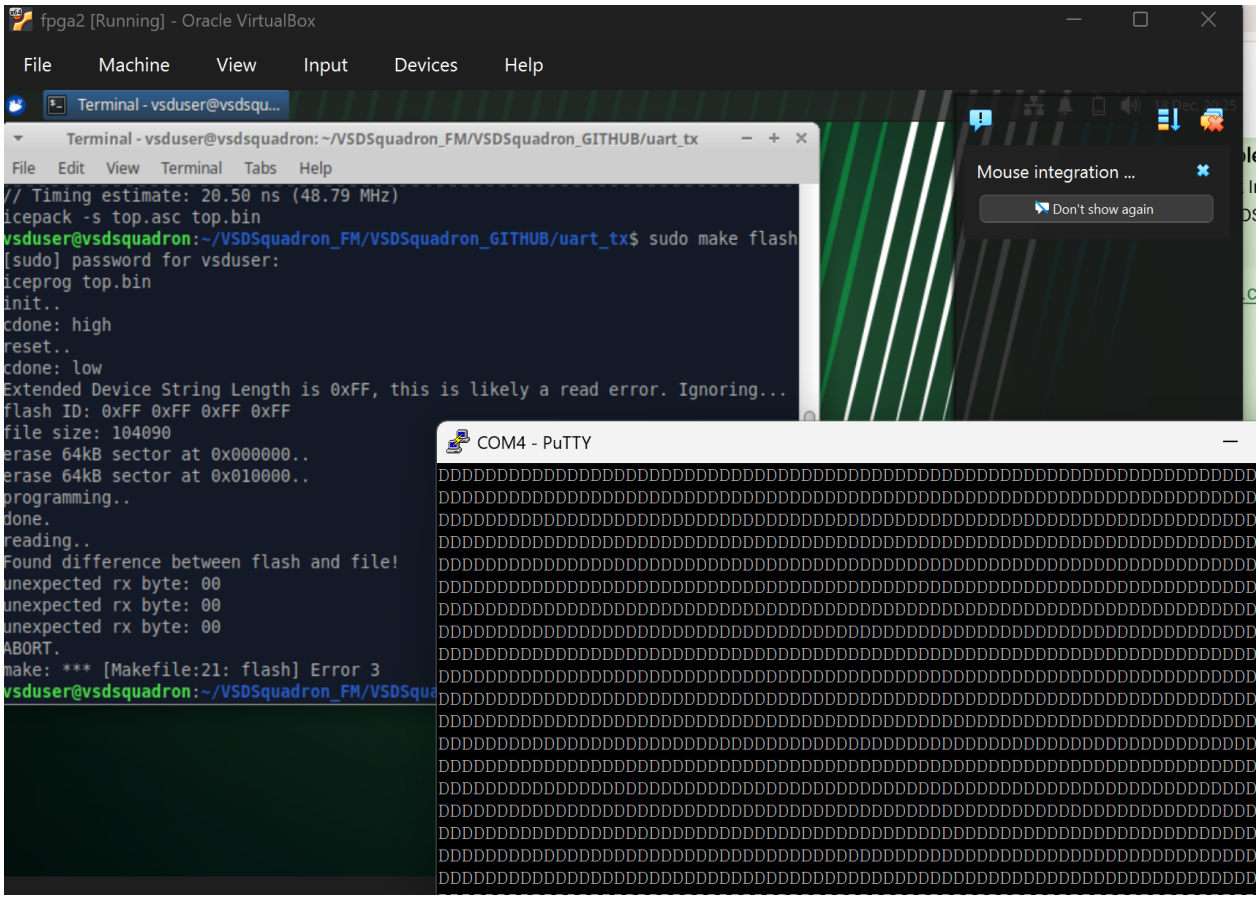
i. After programming the FPGA with the UART transmitter design, the USB-UART adapter was connected to the PC.

ii. The serial terminal software PuTTY was opened, and the correct COM port corresponding to the adapter was selected.
iii. The baud rate was set to 9600 bps, with 8 data bits, no parity, and 1 stop bit (8N1). Flow control was disabled.
iv. Upon starting the terminal session, the ASCII character "D" was continuously transmitted and displayed in the PuTTY window.
v. No data corruption, missing characters, or unexpected symbols were observed, indicating that the UART transmitter was operating correctly.
vi. The continuous stream of "D" matched the periodic trigger derived from the frequency counter, confirming the functionality of the baud-rate generator and the FSM in the UART module.
vii. Additionally, the RGB LEDs on the FPGA board were active, showing dynamic patterns and providing visual confirmation of system operation.

OUTPUT

The FPGA continuously transmitted the character "D" over UART, which was displayed repeatedly on the PuTTY serial terminal.




fpga2 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Terminal - vsduser@vsdsqu...
Terminal - vsduser@vsdsquadron: ~/VSDSquadron_FM/VSDSquadron_GITHUB/uart_tx
File Edit View Terminal Tabs Help
// Timing estimate: 20.50 ns (48.79 MHz)
icepack -s top.asc top.bin
vsduser@vsdsquadron:~/VSDSquadron_FM/VSDSquadron_GITHUB/uart_tx$ sudo make flash
[sudo] password for vsduser:
iceprog top.bin
init..
cdone: high
reset..
cdone: low
Extended Device String Length is 0xFF, this is likely a read error. Ignoring...
flash ID: 0xFF 0xFF 0xFF 0xFF
file size: 104090
erase 64kB sector at 0x000000..
erase 64kB sector at 0x010000..
programming..
done.
reading..
Found difference between flash and file!
unexpected rx byte: 00
unexpected rx byte: 00
unexpected rx byte: 00
ABORT.
make: *** [Makefile:21: flash] Error 3
vsduser@vsdsquadron:~/VSDSqua
Mouse integration ...
Don't show again
COM4 - PuTTY