VHDLwhiz.com

AXI-Style UART Modules User Manual

Version:	1.0.2
Date:	December 20, 2022
Product URL:	https://vhdlwhiz.com/product/vhdl-module-axi-style-uart/
Contact email:	jonas@vhdlwhiz.com

This document describes how to use VHDLwhiz's general-purpose, AXI-compatible, universal asynchronous receiver-transmitter (UART) VHDL modules.

Table of Content

License

The MIT license covers the source code's copyright requirements and terms of use. Refer to the *LICENSE.txt* file in the Zip file for details.

Changelog

These changes refer to the project files, and this document is updated accordingly.

Version	**Remarks**
1.0.0	Initial release

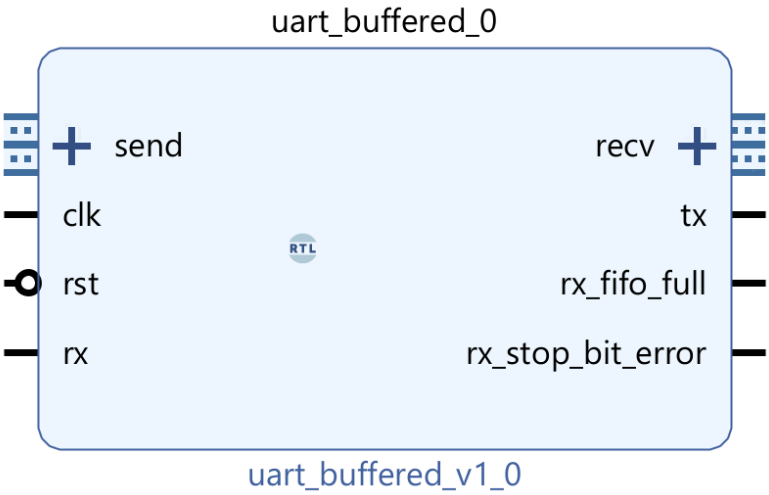
1.0.1	• Created this user manual
	• Replaced self-reset process in the top.vhd file with a separate reset_sync.vhd module
	• Added demo UART loopback projects for these boards:
	  ○ Xilinx Arty A7 35T (Vivado)
	  ○ Xilinx arty S7 50 (Vivado)
	  ○ Terasic DE10-Lite + Digilent Pmod USBUART (Quartus)

1.0.2	• Changing the direction of an internal counter to make sure that the modules also work without an initial reset strobe

Description

This project contains three general-purpose, AXI compatible, universal asynchronous receiver-transmitter (UART) modules.

You can choose between the separate receiver (*uart_rx.vhd*) and transmitter (*uart_tx.vhd*) modules or a combined UART module (*uart_buffered.vhd*) with built-in input and output buffering.



All three modules have built-in flow control through the use of ready/valid handshake signals. And because the signal naming scheme matches the Xilinx AXI standard, the *send* and *recv* ports will appear as bus interfaces in the Vivado Block Design editor, as shown in the image above.

The system clock frequency and the baud rate are configurable. Data bit count is fixed to 8 (one byte), stop bits is set to 1, and parity is not used.

UART RX/TX with block RAM buffering

The *uart_buffered.vhd* module has built-in flow control through the AXI-style read/valid signaling scheme on the send and receive ports. Data transfer only happens when *<send/recv>_tvalid* and *<send/recv>_tready* are '1' during the same rising clock edge.

Buffered bytes are stored in block RAM until they are sent by this module or read from it. The block RAM depth is configurable through generics, but the FIFO size will be that number minus one because the internal FIFO always keeps one slot open to distinguish between the full and empty states.

The listing below shows the entity for this module.

```
entity uart_buffered is
  generic (
    clk_hz : positive;
    baud_rate : positive;
    rx_fifo_ram_depth : positive := 2048;
    tx_fifo_ram_depth : positive := 2048
  );
  port (
    clk : in std_logic;
    rst : in std_logic;

    -- UART input/output lines
    rx : in std_logic;
    tx : out std_logic;

    -- Received byte (sample on pulsed valid)
    recv_tdata : out std_logic_vector(7 downto 0);
    recv_tvalid : out std_logic;
    recv_tready : in std_logic;

    -- Word to send (ready/valid handshake)
    send_tdata : in std_logic_vector(7 downto 0);
    send_tvalid : in std_logic;
    send_tready : out std_logic;

    -- Error conditions ('0' = no error)
    rx_fifo_full : out std_logic;
    rx_stop_bit_error : out std_logic
  );
end uart_buffered;
```

UART RX

The *uart_rx.vhd* module contains an unbuffered UART receiver. Received data appears on the *recv_tdata* line and is valid when *recv_tvalid* is '1'.

The listing below shows the entity for the receiver module.

```
entity uart_rx is
  generic (
    clk_hz : positive;
    baud_rate : positive
  );
  port (
    clk : in std_logic;
    rst : in std_logic;

    -- UART input line
    rx : in std_logic;

    -- Received byte (sample on pulsed valid)
    recv_tdata : out std_logic_vector(7 downto 0);
    recv_tvalid : out std_logic;

    -- Is '1' if the stop bit of the latest byte was '0'
    stop_bit_error : out std_logic
  );
end uart_rx;
```

UART TX

The *uart_tx.vhd* module contains an unbuffered UART transmitter. Data put on *send_tdata* is transmitted when *send_tvalid* is '1'.

The listing below shows the entity for the transmitter module.

```
entity uart_tx is
  generic (
    clk_hz : positive;
    baud_rate : positive
  );
  port (
    clk : in std_logic;
    rst : in std_logic;

    -- UART output line
    tx : out std_logic;

    -- Word to send (ready/valid handshake)
    send_tdata : in std_logic_vector(7 downto 0);
    send_tvalid : in std_logic;
    send_tready : out std_logic
  );
end uart_tx;
```

Zip File Content

.

├── **loopback_demo**	UART loopback demo projects
│ ├── **icecube2_icestick**	iCEcube2 project for the Lattice iCEstick board
│ ├── **quartus_de10_lite**	Quartus project for the Terasic DE10-Lite board
│ ├── **top_sim**	Top-level testbench for the demo project
│ ├── **top_src**	Top-level module for the demo project
│ ├── **vivado_arty_a7_35t**	Vivado project for the Xilinx Arty A7 35T board
│ └── **vivado_arty_s7_50**	Vivado project for the Xilinx Arty S7 50 board
├── **uart_sim**	Testbench for the UART modules
└── **uart_src**	UART VHDL modules

Simulating the design

There are two self-checking testbenches in the Zip file, one for the *uart_buffered* module and one for the generic *top* module used in the demo projects.

The VHDL testbench should work in any capable VHDL simulator supporting the full 2008 VHDL revision, but the provided *run.do* scripts only work in ModelSim/Questa.

Running the UART testbench

Open ModelSim/Questa and type in the simulator console:

```
cd <zip_content>
do uart_sim/run.do
runtb
```

Running the top-level testbench

Open ModelSim/Questa and type in the simulator console:

```
cd <zip_content>
do loopback_demo/top_sim/run.do
runtb
```

Implementing the demo projects

The Zip file contains demo projects for selected development boards.

If you want to try the demo on a different board, you can create a wrapper for the *loopback_demo/top_src/top.vhd* module. Examine the *top_<board_name>.vhd* files and customize one to suit your board's pin configuration and clock frequency.

The example design instantiates the *uart_buffered* module and echoes any bytes it receives over the UART RX pin to the TX pin.

The baud rate is set to 115200, but you can change that by altering the top wrapper VHDL files in each *loopback_demo* subfolder.

Lattice iCEstick

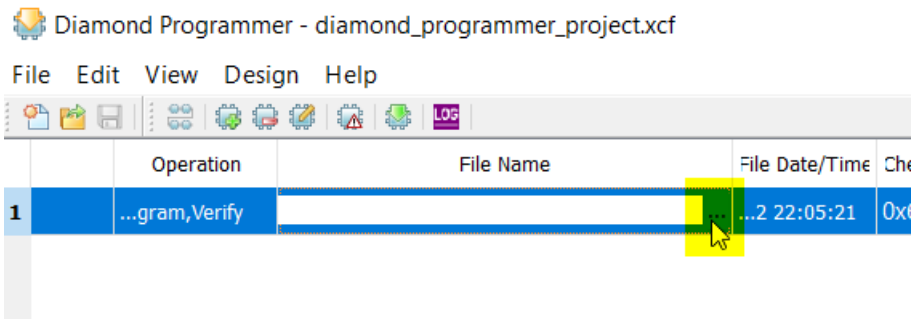
The *loopback_demo/icecube2_icestick* folder contains the loopback implementation for the Lattice iCEstick FPGA board.

To run through the implementation process, open the *loopback_demo/icecube2_icestick/icecube2_project/loopback_demo_sbt.project* file in the Lattice iCEcube2 design software.

After loading the project in the iCEcube2 GUI, click **Tools→Run All** to generate the programming bitmap file.

You can use the Lattice Diamond Programmer Standalone tool to configure the FPGA with the generated bitmap file. When Diamond Programmer opens, click **Open an existing programmer project** in the welcome dialog box.

Select project file found in the Zip:
loopback_demo/icecube2_icestick/diamond_programmer_project.xcf and click OK.



After the project loads, click the three dots in the **File Name** column, as shown above. Browse to select the bitmap file that you generated in iCEcube2:

loopback_demo/icecube2_icestick/icecube2_project/loopback_demo_Implmnt/sbt/outputs/bitmap/top_icestick_bitmap.bin

Finally, with the iCEstick board plugged into a USB port on your computer, select **Design→Program** to program the SPI flash and configure the FPGA.

You can now proceed to test the loopback design using a serial terminal.

Xilinx Arty A7 35T

You can find the demo implementation for the Artix-7 35T Arty FPGA evaluation kit in the *loopback_demo/vivado_arty_a7_35t* folder.

Open Vivado and navigate to the extracted files using the Tcl console found at the bottom of the GUI interface. Type this command to enter the demo project folder:

```
cd <zip_content>/loopback_demo/vivado_arty_a7_35t/
```

Execute the *create_vivado_proj.tcl* Tcl script to regenerate the Vivado project:

```
source ./create_vivado_proj.tcl
```

Click **Generate Bitstream** in the sidebar to run through all the implementation steps and generate the programming bitstream file.

Finally, click **Open Hardware Manager** and program the FPGA through the GUI. The bitfile should be in this folder:
<zip_content>/loopback_demo/vivado_arty_a7_35t/loopback_demo.runs/impl_1/top_arty_a7_35t.bit

You can now proceed to test the loopback design using a serial terminal.

Xilinx Arty S7 50

You can find the demo implementation for the Arty S7: Spartan-7 FPGA development board in the *loopback_demo/vivado_arty_s7_50* folder.

Open Vivado and navigate to the extracted files using the Tcl console found at the bottom of the GUI interface. Type this command to enter the demo project folder:

```
cd <zip_content>/loopback_demo/vivado_arty_s7_50/
```

Execute the *create_vivado_proj.tcl* Tcl script to regenerate the Vivado project:

```
source ./create_vivado_proj.tcl
```

Click **Generate Bitstream** in the sidebar to run through all the implementation steps and generate the programming bitstream file.

Finally, click **Open Hardware Manager** and program the FPGA through the GUI. The bitfile should be in this folder:
<zip_content>/loopback_demo/vivado_arty_s7_50/loopback_demo.runs/impl_1/top_arty_s7_50.bit

You can now proceed to test the loopback design using a serial terminal.

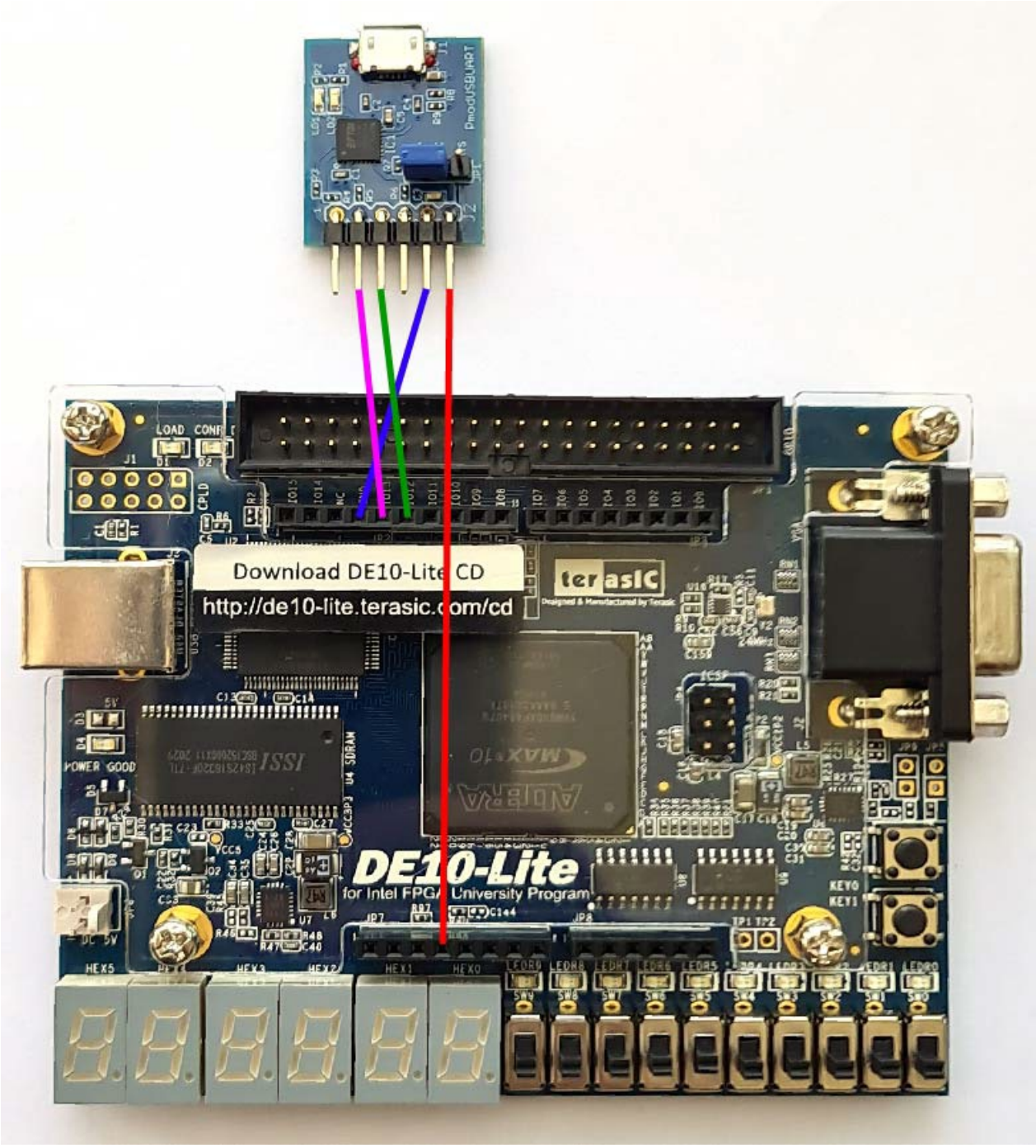
Terasic DE10-Lite with Digilent Pmod USBUART

The loopback demo for the DE10-Lite board from Terasic, using Intel's MAX 10 FPGA, is in the *loopback_demo/quartus_de10_lite* folder.

Because the DE10-Lite board doesn't have a built-in UART to USB interface, we will use Digilent's pluggable USBUART Pmod module for that.

DE10-Lite header pin	**Pmod USBUART pin**
Arduino header IO13	2 RXD
Arduino header IO12	3 TXD
Arduino header GND	5 GND
Arduino header VCC3P3	6 VCC

Connect the DE10-Lite and Pmod according to the pin mappings in the table above, as shown in the image below.



Open the project file in Quartus to load the design:

<zip_content>/loopback_demo/quartus_de10_lite/loopback_demo.qpf

Click **Compile Design** from the sidebar

Click **Program Device (Open Programmer)** from the sidebar

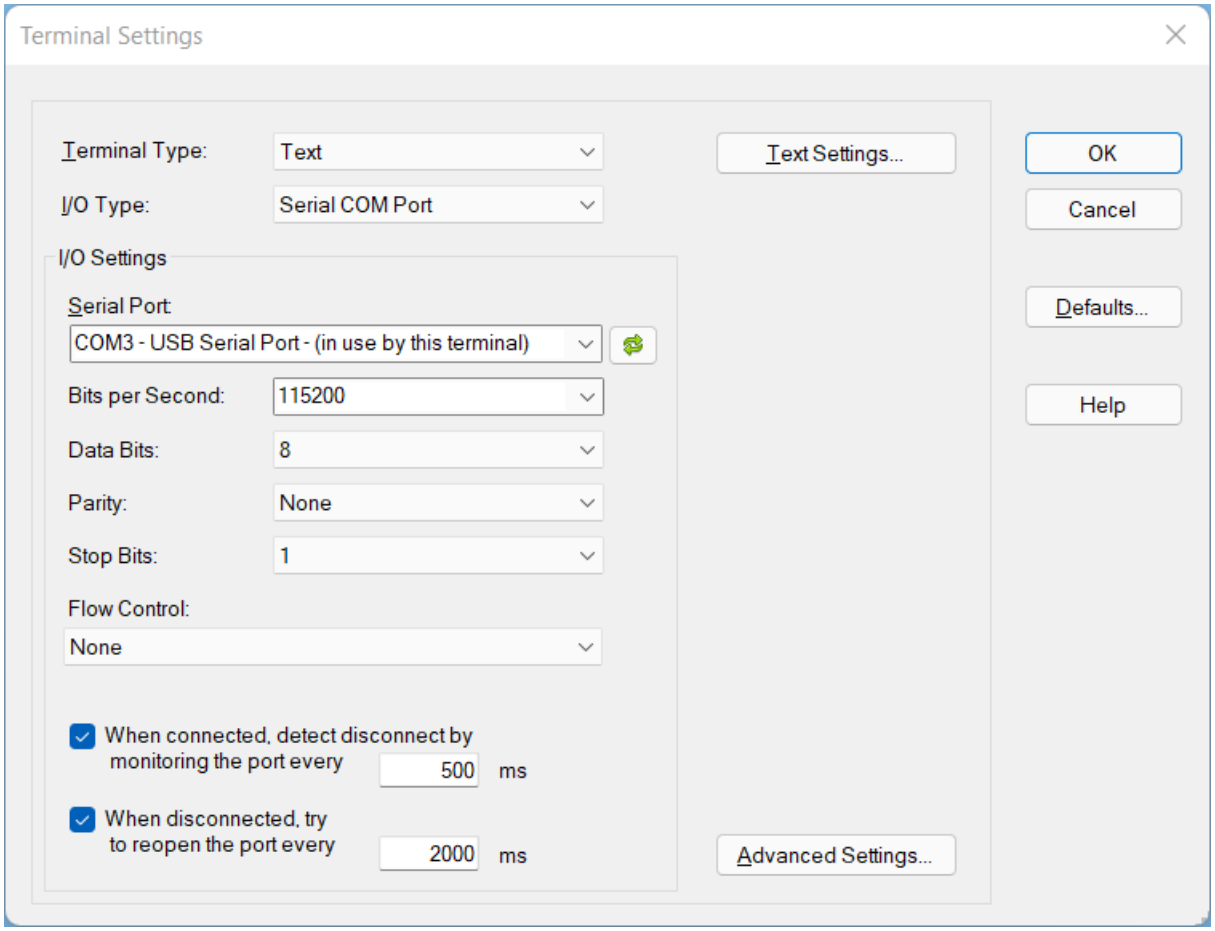
The programming file should be selected automatically. You can find it at:

<zip_content>/loopback_demo/quartus_de10_lite/output_files/loopback_demo.sof

Testing the UART loopback

You can use any serial terminal program to communicate with the board, but I recommend YAT – Yet Another Terminal if you don't already have one installed.

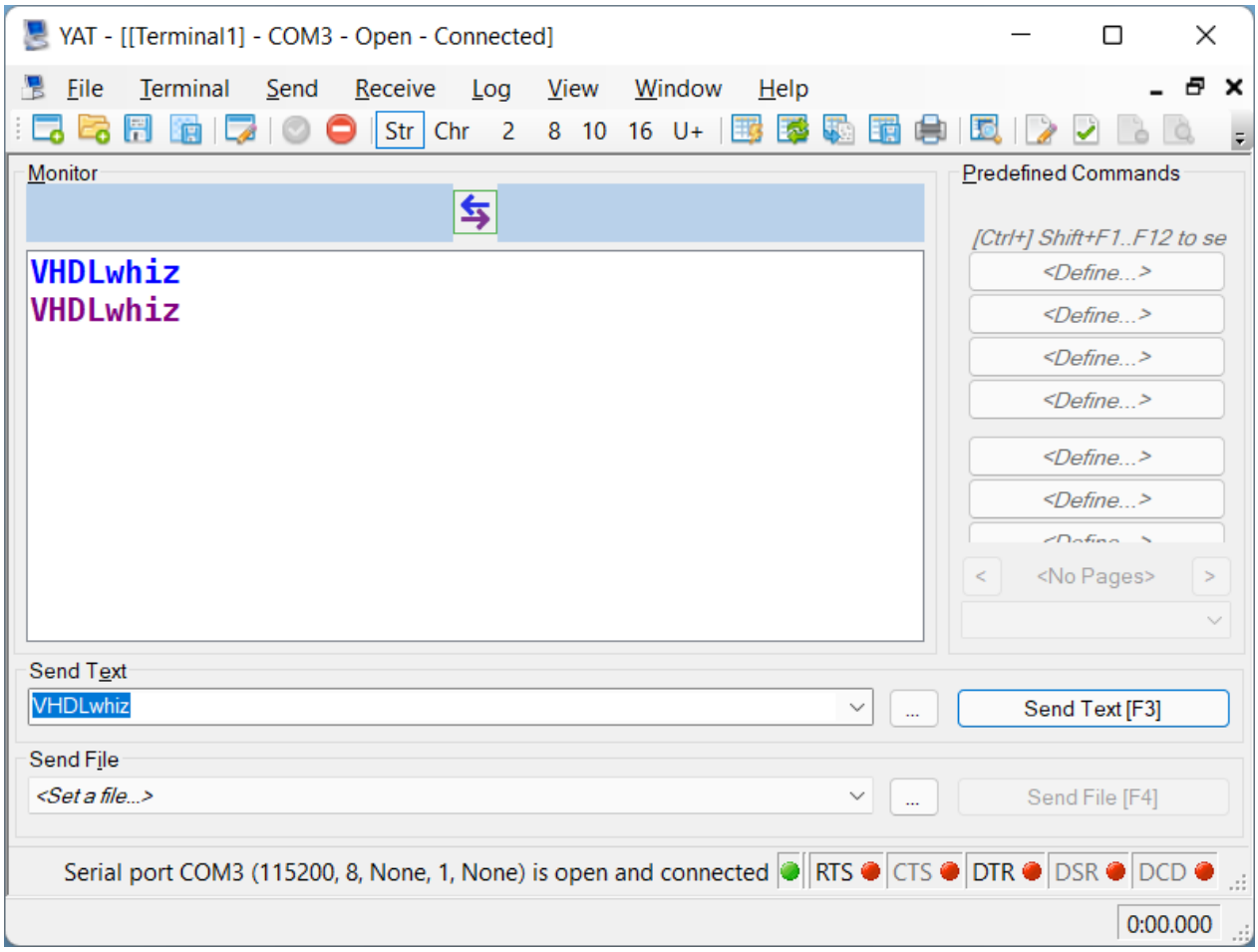
Go to **Terminal→Settings** and set the baud rate to 115200, data bits to 8, parity to none, and stop bits to 1, as shown in the screenshot below.



Connect the board or Pmod UART module to your computer over USB.

Finally, select **Terminal→Open/Start**, enter some characters in the **Send Text** field and hit the **Send Text** button. The sent text will appear in blue, and if the FPGA responds, the echoed text will appear once more below in purple color.

You can see an example of a successful test in the screenshot below.



Known Issues

The UART transmitter module does not work correctly in with the Yosys open-source synthesis suite. The *clk_counter_wrapped* impure function in the *uart_tx.vhd* file doesn't synthesize as intended in this tool.