**1** Sign in to follow this

Sign Up

A National Instruments Company
ects Documentation Projects

Search... Leaderboard **Browse** Activity

Q

**Followers** 

**Forums** 

Gallery

Guidelines

Staff

Online Users

★ Home > Digilent Technical Forums > FPGA > VHDL UART RX for Nexys4 DDR?

All Activity

1





# VHDL UART RX for Nexys4 DDR?





Asked by DoctorWkt



## Question

### **DoctorWkt**

Member



Members **O** 3 20 posts

Posted July 7, 2015 (edited)

HI, I got my Nexys4 DDR about 3 weeks back, and I've been able to bring up my own microcoded CPU on the board. Lots of fun! I borrowed the UART transmit code from one of the examples in the Resource Center. Is there an example with UART receive code in it, or does anybody have a simple 9600,8,N,1 UART receive component which would work?

Many thanks, Warren

Edited July 9, 2015 by JColvin

SORT BY DATE SORT BY VOTES

hamster

Overly helpful

Posted July 8, 2015









 I did find some UART receive code and fixed a few bugs in it. I've written a VHDL project where the UART reads a character and echoes it back. However, at present I am losing about half the characters that I type with minicom on Linux.

Attached is a Zip with the project.

UART\_TX\_CTRL.vhd comes from the Resource
Center, UART\_RX\_CTRL.vhd is the receive code.
serialport.vhd is the top-level file to generate the
Nexys4 bitstream. There are also behavioural
simulation testbeds uart\_tx\_test.vhd and
uart\_rx\_test.vhd.

I'm still a VHDL newbie, but if someone could help me get this work then it could be the basis for a project to put into the Resource Center. Right now I have three inferred latches, and I'm not sure what I need to do to fix them.

Any help would be much appreciated! Warren

serialport.zip

#### Hi Warren,

Here's a nasty/inefficient implementation of a UART that should be relatively bulletproof, as long as there is minimal noise on the line.

```
READ_DATA: out STD_LOGIC :=
           RESET_READ: in STD_LOGIC
    );
end UART_RX_CTRL;
architecture behavioral of UART_RX_CTRL i
   constant FREQ : integer := 100000000;
   constant BAUD : integer := 9600;
   -- Needs to hold the count of 9.5 bit-
   signal
            count : unsigned(16 downt
   constant sample_0 : unsigned(16 downt
   constant sample_1 : unsigned(16 downt
   constant sample_2 : unsigned(16 downt
   constant sample_3 : unsigned(16 downt
   constant sample_4 : unsigned(16 downt
   constant sample 5 : unsigned(16 downt
   constant sample_6 : unsigned(16 downt
   constant sample 7 : unsigned(16 downt
   constant stop_bit : unsigned(16 downt
   signal
           byte : std_logic_vector(
begin
rx_state_process : process (CLK)
   begin
      if(rising_edge(CLK)) then
         READ_DATA <= '0';</pre>
         case count is
            when sample_0 => byte <= UART
            when sample_1 => byte <= UART
           when sample_2 => byte <= UART
            when sample_3 => byte <= UART
           when sample_4 => byte <= UART
            when sample 5 => byte <= UART
            when sample 6 => byte <= UART
           when sample_7 => byte <= UART
           when stop bit =>
               -- Send out the data when
               if UART RX = '1' then
                  DATA <= byte;
                  READ DATA <= '1';
               end if;
```

If using it in an environment where noise might be a problem you count first filter UART\_RX (like denouncing a switch), but other ways are better.

The best way might be to sample the signal at five times the baud rate, and when you see four '0's in a row then wait for three sample intervals (to get to the middle of the first data bit, then sample bits every five sample intervals. It sounds a lot more complex then it is to implement, depending on if you go for a long shift register (code untested, just to illustrate the idea):

```
...
process(clk)
begin
if rising_edge(clh) then
if count = FREQ/BAUD/5 then
count <= (others => '0');
```

```
shift_reg <= UART_RX & shift_reg(shift_reg'high
downto 1);
  else
   count <= count +1;
  end if;
  if shift_reg(3 downto 0) = "0000" then -- start bit
at the head of the register
   DATA(0) \le shift_{reg}(7);
   DATA1) <= shift_reg(12);
   DATA(2) <= shift_reg(17);
   DATA(3) \le shift_reg(22);
   DATA(4) \le shift_reg(27);
   DATA5) <= shift_reg(32);
   DATA(6) \le shift_reg(37);
   DATA(7) \le shift_reg(42);
   READ_DATA <= shift_reg(47); -- use the stop bit
to assert the data strobe (U)
   shift_reg <= (others =>'1'); -- empty out the shift
register.
  else
   READ_DATA <= '0';
  end if;
end if
end process;
```



JColvin Forum Moderator



Administrators

\*\* Staff

• 270

2824 posts

LocationPullman, WA

Posted July 7, 2015

Hi Warren,

Your project definitely looks pretty cool!

I'm not aware of any UART receive code on our wiki, although we would like to get some up there in the future. For now, I would probably check out this RS232 serial interface walkthrough <a href="here">here</a> (as recommended to me by another user).

Let me know if you have any more questions.

Thanks, JColvin



## DoctorWkt Member





Members

3

20 posts

Posted July 8, 2015

I did find some UART receive code and fixed a few bugs in it. I've written a VHDL project where the UART reads a character and echoes it back. However, at present I am losing about half the characters that I type with minicom on Linux.

Attached is a Zip with the project.

UART\_TX\_CTRL.vhd comes from the Resource Center, UART\_RX\_CTRL.vhd is the receive code. serialport.vhd is the top-level file to generate the Nexys4 bitstream. There are also behavioural simulation testbeds uart\_tx\_test.vhd and uart\_rx\_test.vhd.

I'm still a VHDL newbie, but if someone could help me get this work then it could be the basis for a project to put into the Resource Center. Right now I have three inferred latches, and I'm not sure what I need to do to fix them.

Any help would be much appreciated! Warren

serialport.zip



## **DoctorWkt**

Member 00



Members **3** 20 posts

#### Posted July 8, 2015

Hamster gave me a good tip about the inferred latches. I have fixed up the code and added more comments. Now I can transmit a text file down at 9600bps and most of the characters come back. But a few of them are being corrupted. I've checked the states in the receiver and they all seem fine. The corruption is not predictable, i.e. not every N characters are corrupted when I am blasting a text file at 9,600,8,N,1.

Attached is the latest version of the code in case someone wants to chew on it :-)

Cheers, Warren

serialport\_v2.zip



## **DoctorWkt**

Member 



Members **O** 3 20 posts

#### Posted July 8, 2015

Yet more data. I created a component to make it easier to display hex data on the 7-segment digits, see attached file. Might be useful in other projects. I'm using it to display the last four characters in hex. Using this, I can see that the data is being corrupted in the UART receive code and not in the UART transmit code.

Cheers, Warren

hex\_7seq.vhd



<

<



#### On 7/7/2015 at 11:46 PM, DoctorWkt said:

I did find some UART receive code and fixed a few bugs in it. I've written a VHDL project where the UART reads a character and echoes it back. However, at present I am losing about half the characters that I type with minicom on Linux.

Attached is a Zip with the project.

UART\_TX\_CTRL.vhd comes from the Resource
Center, UART\_RX\_CTRL.vhd is the receive code.
serialport.vhd is the top-level file to generate the
Nexys4 bitstream. There are also behavioural
simulation testbeds uart\_tx\_test.vhd and
uart\_rx\_test.vhd.

I'm still a VHDL newbie, but if someone could help me get this work then it could be the basis for a project to put into the Resource Center. Right now I have three inferred latches, and I'm not sure what I need to do to fix them.

Any help would be much appreciated! Warren

serialport.zip

Hi Warren,

Here's a nasty/inefficient implementation of a UART that should be relatively bulletproof, as long as there is minimal noise on the line.

```
-- UART_RX_CTRL.vhd -- Simple UART RX

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity UART_RX_CTRL is
   port ( UART_RX: in STD_LOGIC;
```

```
CLK:
                       in
                           STD_LOGIC;
           DATA:
                       out STD_LOGIC_VEC
           READ_DATA: out STD_LOGIC :=
           RESET_READ: in
                           STD_LOGIC
    );
end UART_RX_CTRL;
architecture behavioral of UART_RX_CTRL i
  constant FREQ : integer := 100000000;
  constant BAUD : integer := 9600;
   -- Needs to hold the count of 9.5 bit-
           count : unsigned(16 downt
  signal
  constant sample_0 : unsigned(16 downt
  constant sample_1 : unsigned(16 downt
  constant sample_2 : unsigned(16 downt
  constant sample_3 : unsigned(16 downt
  constant sample_4 : unsigned(16 downt
  constant sample 5 : unsigned(16 downt
  constant sample_6 : unsigned(16 downt
  constant sample_7 : unsigned(16 downt
  constant stop bit : unsigned(16 downt
           byte : std_logic_vector(
  signal
begin
rx_state_process : process (CLK)
  begin
      if(rising_edge(CLK)) then
        READ_DATA <= '0';</pre>
        case count is
           when sample_0 => byte <= UART
           when sample_1 => byte <= UART
           when sample_2 => byte <= UART
           when sample 3 => byte <= UART
           when sample 4 => byte <= UART
           when sample_5 => byte <= UART
           when sample 6 => byte <= UART
           when sample_7 => byte <= UART
           when stop bit =>
              -- Send out the data when
              if UART_RX = '1' then
                 DATA <= byte;
```

```
READ_DATA <= '1';</pre>
                end if;
             when others =>
                null:
         end case;
         if count = stop_bit then
             count <= (others => '0');
         elsif count = 0 then
             if UART_RX = '0' then -- Star
                count <= count + 1;</pre>
             end if;
         else
              count <= count + 1;</pre>
         end if;
      end if;
   end process;
end behavioral;
[/code]
```

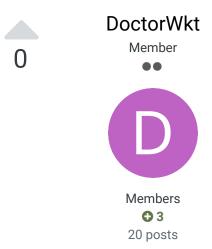
If using it in an environment where noise might be a problem you count first filter UART\_RX (like denouncing a switch), but other ways are better.

The best way might be to sample the signal at five times the baud rate, and when you see four '0's in a row then wait for three sample intervals (to get to the middle of the first data bit, then sample bits every five sample intervals. It sounds a lot more complex then it is to implement, depending on if you go for a long shift register (code untested, just to illustrate the idea):

```
process(clk)
begin
if rising_edge(clh) then
  if count = FREQ/BAUD/5 then
    count <= (others => '0');
    shift_reg <= UART_RX & shift_reg(selse)</pre>
```

```
count <= count +1;</pre>
     end if;
     if shift_reg(3 downto 0) = "0000" tr
      DATA(0) <= shift_reg(7);</pre>
      DATA(1) <= shift_reg(12);</pre>
      DATA(2) <= shift_reg(17);</pre>
      DATA(3) <= shift_reg(22);</pre>
      DATA(4) <= shift_reg(27);
      DATA(5) <= shift_reg(32);</pre>
      DATA(6) <= shift_reg(37);</pre>
      DATA(7) <= shift_reg(42);</pre>
      READ_DATA <= shift_reg(47); -- use</pre>
      shift_reg <= (others =>'1'); -- emp
    else
      READ DATA <= '0';
    end if;
  end if
end process;
```

Edited July 8, 2015 by hamster



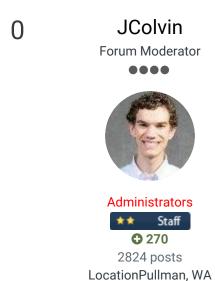
Posted July 9, 2015

Thanks Hamster. That code is much easier to read and it works! I have tidied it up a bit and added some more comments. Attached is a working project that demonstrates the UART receive and transmit functionality on the Nexys4 (DDR). Would it be possible for Digilent to repackage this as a wiki entry on their Resource Center for the Nexys4 DDR?

Cheers & thanks all, Warren

serialport\_v3.zip

<



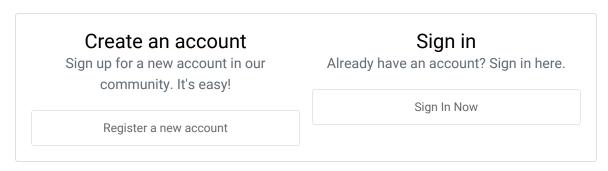
We can definitely incorporate the project into our Wiki!

I'll inform the appropriate people and they'll get hopefully get that uploaded in the near future.

Thank you for helping us out!

## Create an account or sign in to comment

You need to be a member in order to leave a comment







All Activity



Privacy Policy Contact Us

Copyright Digilent Inc. 2018
Powered by Invision Community