Minimal UART core

Author: Arao Hayashida Filho – Published on *opencores.org*

1- Introduction

The fundamental idea of this core is implement a very simple UART in VHDL, using less quantity of logic resources, and a core that can fit on a small CPLD.

For this implementation the core only have an assyncronous receiver, transmitter and a baud rate generator, that can be configured changing a constant value on the baud rate generator component.

One example of application is the test system that were made, a 16x2 character LCD display based on HD44780 or KS0070b, being interfaced with the EIA-232 protocol.

On the development was used the version 10.1 of the Xilinx ISE webpack (syntesis on XST), with simulations done on the Modelsim XE III 6.3c, free versions avaiable on Xilinx downloads.

The hardware tests were done on a Spartan 3E FPGA ,XC3s100e , one seven segment led display and a MDLS162S65SS-01 LCD, with the KS0070b controller.

2- Licensing

All the project files were published on the LGPL terms, you must read the GNU Lesser General Public License for more details.

3- Description of operation

a-Baud rate generator:

The principal clock, on the test boar was 40MHz, this clock will be the clock of a counter, that when reaching some fixed value will generate one pulse on the output of the comparator.

The core is full duplex, so it has two clocks, one for receiver and another for the transmitter section.

For reasons of synchronizing receiver signals, the counter must be loaded with a value after the detect of a start bit condition, such value will be the half of the baud value, because this will be the best value without receiving errors on a wide range of the transmitting baud rate errors.

For the transmitter baud rate generator, the counter only have a reset, that will be unasserted when a write condition occur.

All the baud rate generator was implemented being synchronous with the CLOCK signal.

If the application don't need the sixteen bits of the counters, change the DIVIDER_WIDTH generic, choosing a lower value will decrease the number of registers, using less resources of the CPLD/FPGA.

The following timing simulation of the baud rate generator component can show how it works.

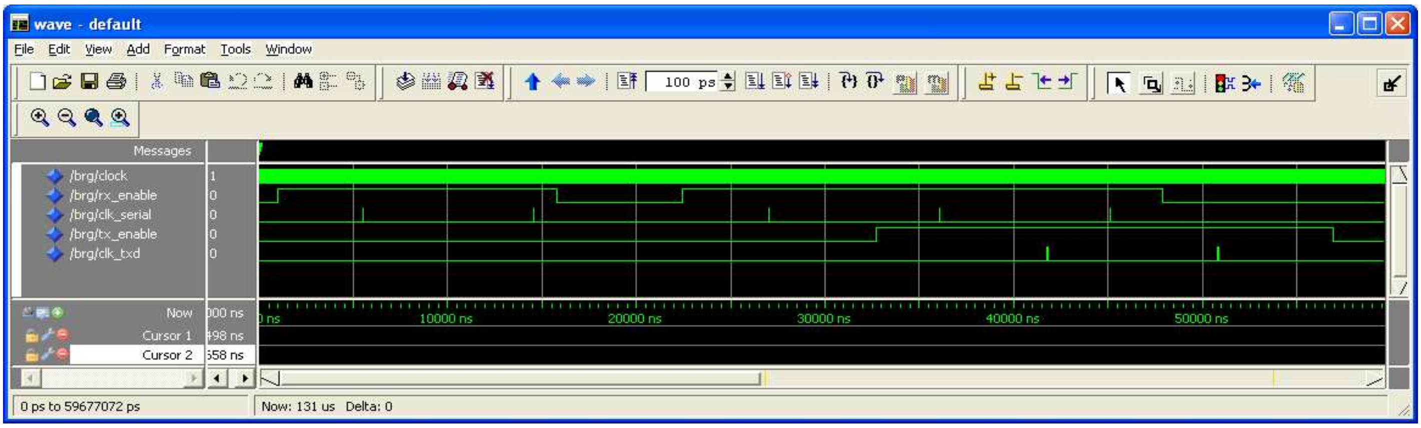


Figure 1 – Baud rate generator post route simulation

On the next simulation screenshot the period of the CLOCK signal is 26ns.

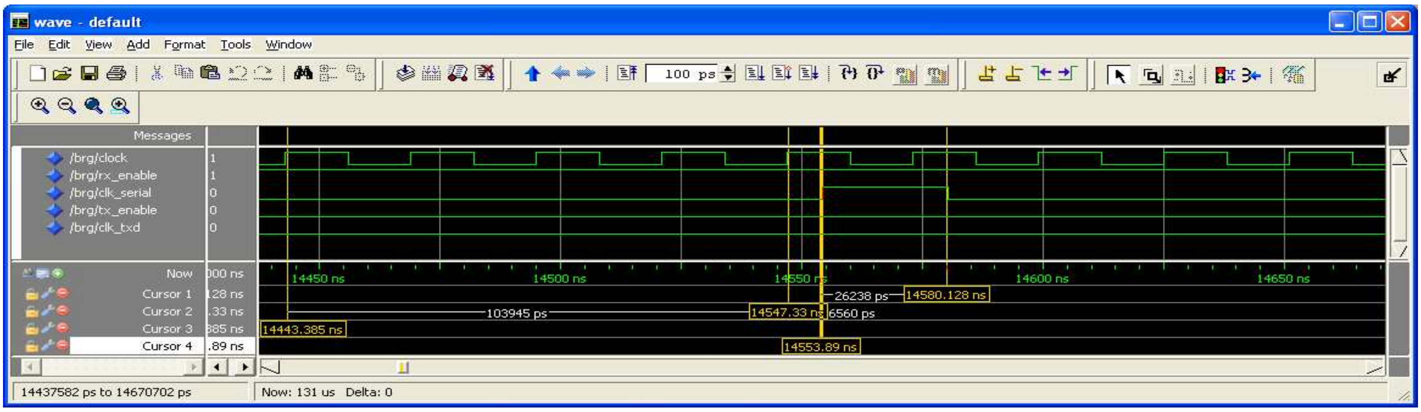


Figure 2 – Zoom of a single CLK_SERIAL pulse

To calculate the BRDVD constant use the following expression:

$$BRDVD = \frac{f_{CLOCK}}{BAUD}$$

For example: for a 1200 baud on a 40MHz clock the divider will be BRDVD=33333 or X”8235”

b-<u>Receiver</u>:

Representing the start, stop and serial data as states, was implemented a machine state named RXD_STATE_MACHINE, that cause the generation of the control signals RECEIVING and EOCS and NEXT_STATE.

If there is start condition, the state machine will do the states S0-S9 and then S0 clocked by the CLK_SERIAL, that have the same period as the bit rate, the RXD_STATES process will load the NEXT_STATE on the ATUAL_STATE signal.

All the receiver shift register is implemented on the RXD_SHIFT process, that will avoid shift on a stop bit condition.

On the end of reception, but after the shift of the last data bit the signal EOC goes low to high, indicating that on the OUTP bus there is valid data, the width of the EOC is the same of the baud rate period.

The following simulation can show the relationship between the baud rate clock and the signals of the receiver when receiving two bytes, the output CLOCK_RXD is equal to the CLK_SERIAL.

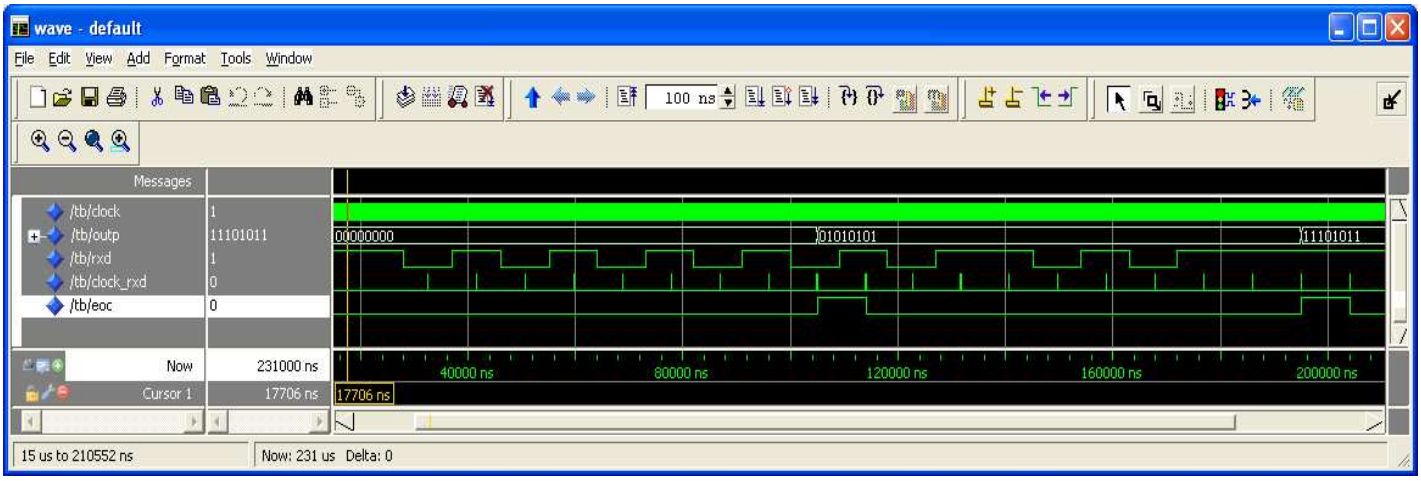


Figure 3 – Post route simulation of the receiver section

c- Transmitter:

The transmission has the same type of state machine, which each states of the TXD_STATE_MACHINE, represent idle/stop, start or data transfer modes.

On the rising edge of the WR signal (TX_ENABLE goes to high logic level), the data on INP bus will be latched to the transmitter register, then, the bits will be trasnsfered to the TXD pin until the machine reach the stop bit (state S9).

Once reaching the S9 state, EOT signalize that the transmitter send the data, and then READY goes to high level, indicating that the transmitter can receive a new data byte.

The next post-route simulation shows how the data can be readed from the core.

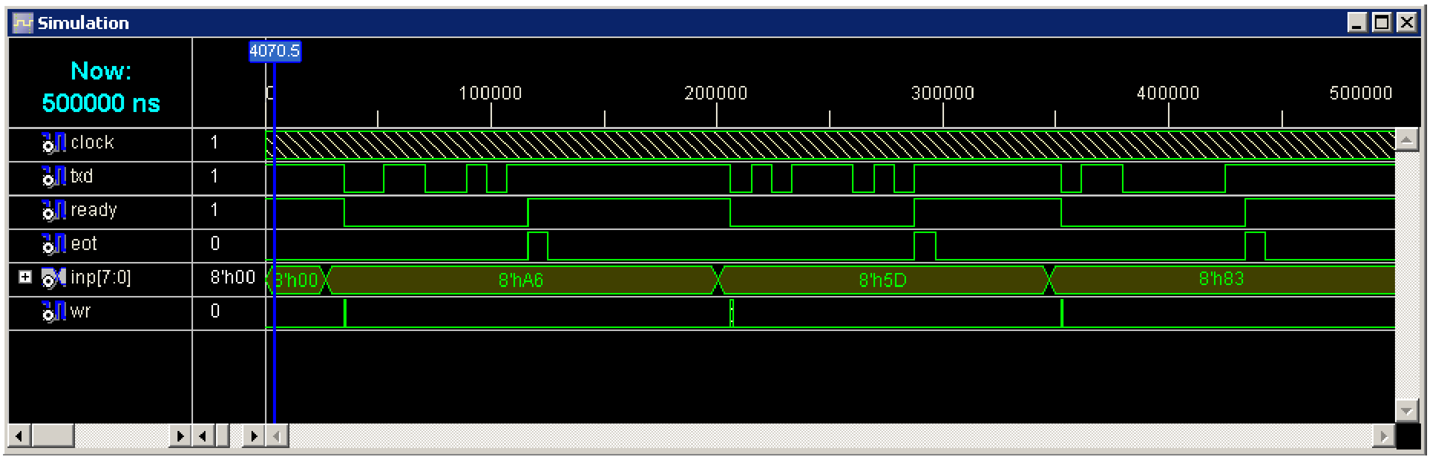


Figure 4 – Post route simulation of the transmitter section

d- Overview:

The next simulation shows which signals and how the core signals change data between the UART and the target system.

For this simulation, the VHDL testbench can be found on the project sources.

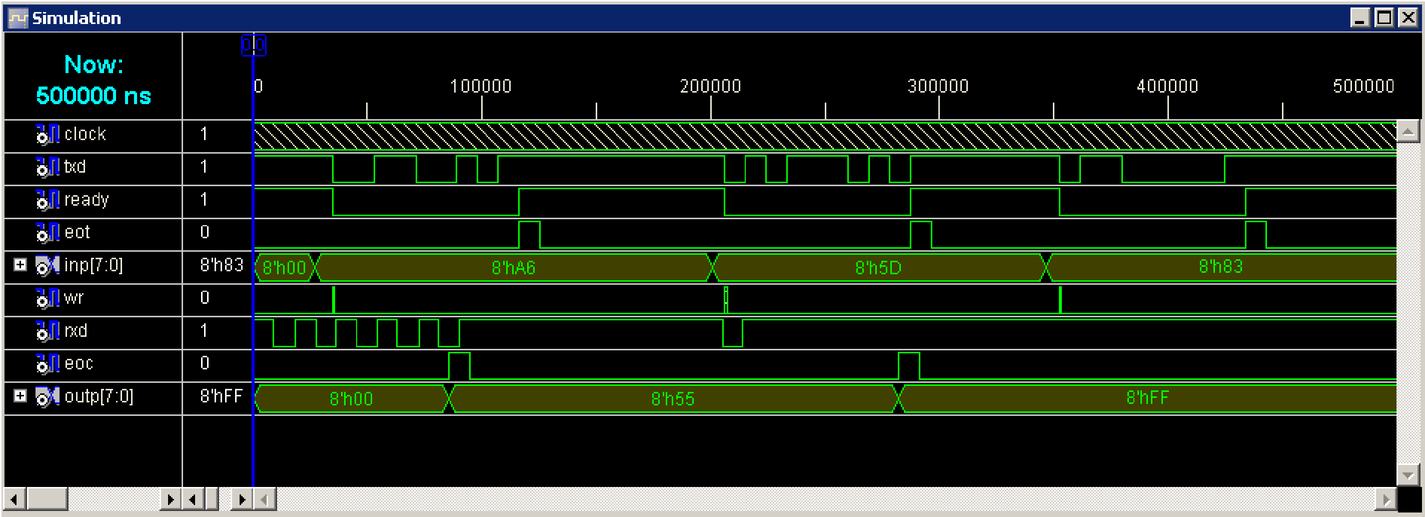
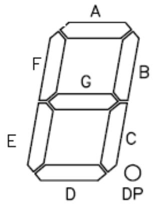


Figure 5 – Complete system simulation

4-Interfacing examples

The first test implementation of this core used a green seven segment led display, cathode common connected by to the fpga pins by eight resistors of 680Ω, with all of the FPGA power pins at 2,5V.

On the next table there is how the display segment are named on the output port.

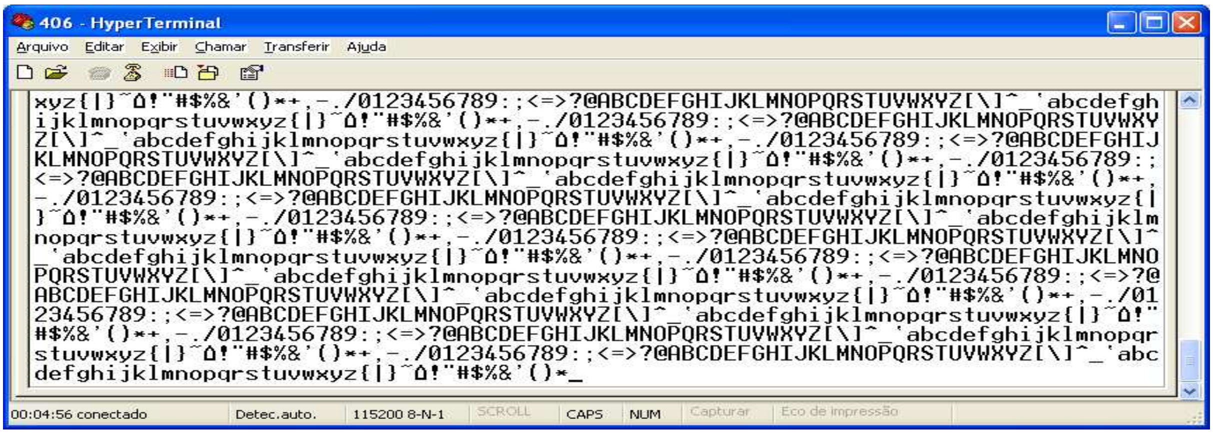


Segment	Signal
A	`SEGMENT(1)`
B	`SEGMENT(2)`
C	`SEGMENT(4)`
D	`SEGMENT(5)`
E	`SEGMENT(6)`
F	`SEGMENT(0)`
G	`SEGMENT(7)`
Dot Point	`SEGMENT(3)`

All the code of this implementation are on the *decod.vhd* file, that contains the seven segment decoder, showing the data received on hexadecimal format.

There is only one display, so the first number will be the most significant four bits and the second number, that has the dot point always on will be the four least significant bits.

For the transmitter section, will be send one byte for each 500*ms* ,as a sequence from the X”21” to X”7F”, as shown on the following terminal screen.



```
xyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
ijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,
-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|
}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
nopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^
_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*+,-./01
23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~⌂!"
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
stuvwxyz{|}~⌂!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abc
defghijklmnopqrstuvwxyz{|}~⌂!"#$%&'()*_
```

Figure 5 – Test of the transmitter

For the next interfacing example was used a common character lcd with the controller Samsung KS0070b, 16 characters, two lines.

The output protocol of the input UART is the same of the lcd and it has better timing specification as informed on the datasheet, except for the RS pin, that set between instruction mode and DDRAM write mode.

To resolve this issue was implemented a process that can change the RS bit receiving a control character on the serial uart component.

At any time, sending the character \ or 92 (decimal), the RS pin goes to low (instruction mode) or sending **]** or 93 (decimal) will set RS pin to high level (DDRAM write mode).

The inicialization codes for the KS0070B controller that must be sended to the LCD are (decimal values):

055,015,006,001,003

That initialization codes set the LCD to the mode: 2 lines, 5x7 font, cursor on, display on, cursor blinking, after sending these commands and changing the data to the DDRAM, ASCII characters can be viewed on the display.

5-Description of the source files

- **-BRG.VHD:** The baud rate generator
- **-DECOD.VHD:** Sven segment display encoder and transmitter test
- **-SERIAL.VHD**: Transmitter and receiver sections
- **-MAIN.VHD:** Implementation of the character LCD
- **-TBW.VHW :** VHDL testbench file

6-Revision history

REV	DATA	Changes
0.1	07/03/2009	First Opencores release

7-References

Datasheet: KS0070 16COM / 80SEG DRIVER & CONTROLLER FOR DOT MATRIX LCD - Samsung semiconductor

Xilinx synthesis and simulation design guide:
http://toolbox.xilinx.com/docsan/xilinx10/books/docs/sim/sim.pdf

Xilinx Synthesis Technology User Guide
http://www.xilinx.com/itp/xilinx5/pdf/docs/xst/xst.pdf