UART LIFO

Andrea Ćirić 2016/202
University of Belgrade, School of Electrical Engineering

Project 16. UART LIFO

USB on UART TTL, 4 × switch

Implement serial transmitter and receiver UART that can run on multiple bit rates. Bit rates are defined using three switches. Computer sends multiple data until carriage return when data should be sent to output in reverse order (Last In First Out).

components

Bit Rate Generator

Generates a bit rate array based on values of three input switches.

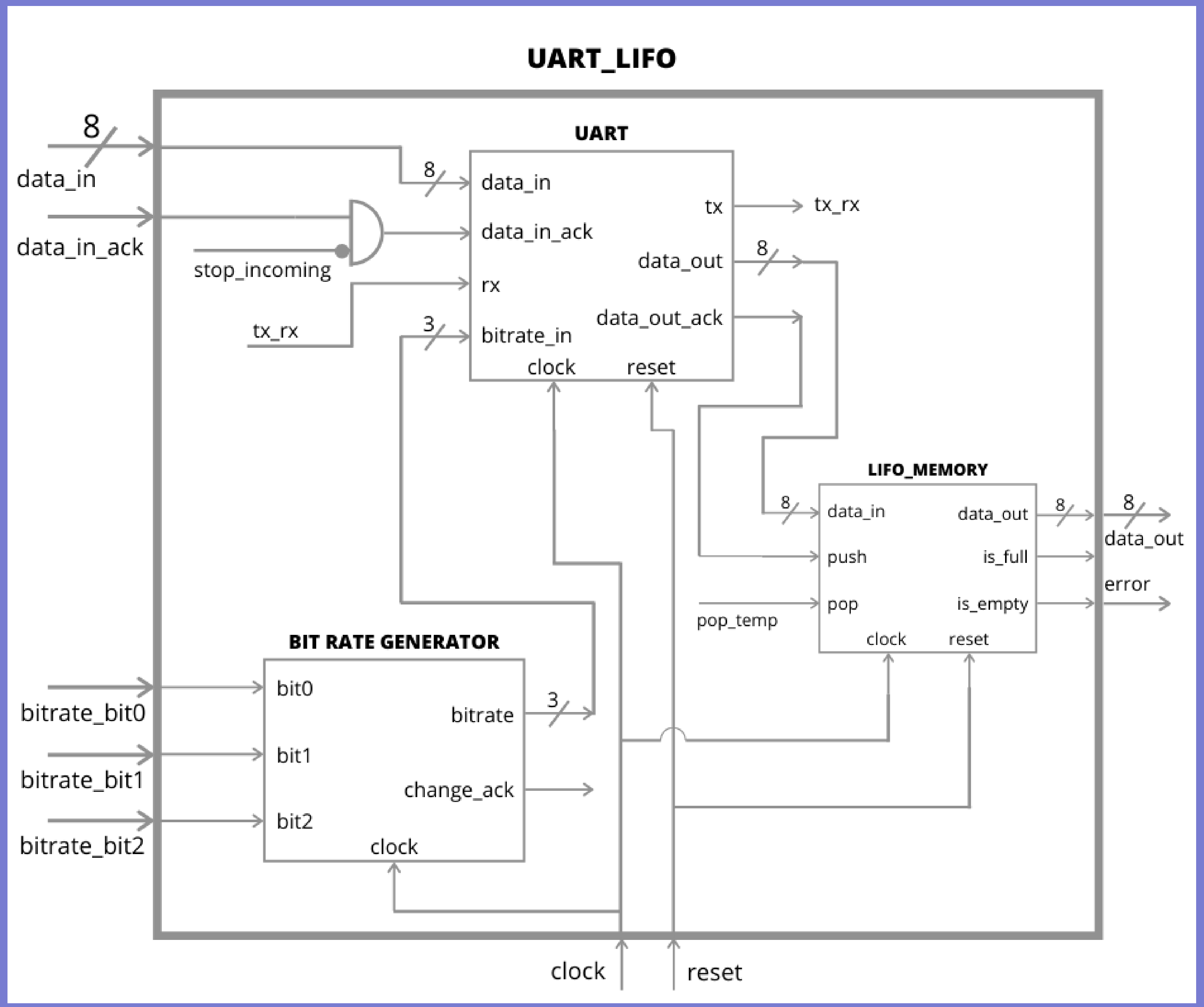
UART

Implementation of UART (Universal Asynchronous Receiver-Transmitter) which can work on 8 different bit rates.

LIFO MEMORY

Implementation of memory that works on the principle of LIFO (Last In First Out) -> stack

components


UART_LIFO
UART
data_in
data_in_ack
8
stop_incoming
tx_rx
rx
3
bitrate_in
clock
reset
tx
tx_rx
data_out
data_out_ack
LIFO_MEMORY
data_in
push
pop
pop_temp
clock
reset
data_out
is_full
is_empty
data_out
error
BIT RATE GENERATOR
bit0
bit1
bit2
bitrate
change_ack
clock
bitrate_bit0
bitrate_bit1
bitrate_bit2
clock
reset

Bit Rate Generator

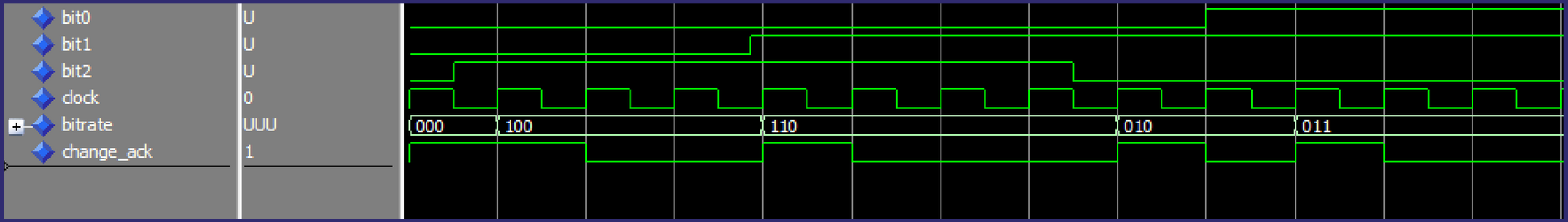
Description:

Gets three input logic values (bit0, bit1, bit2) and depending on them, it generates a three-bit output (bitrate) that goes to the uart component that generates a bitrate (bit/s) value.

Inputs/Outputs:

clock	- system clock
bit0, bit1, bit2	- input values that generate bitrate value
bitrate	- output array generated from input bitX values
change_ack	- 1 if there is a change, otherwise it's 0

Bit Rate Generator



The figure shows how after the change on the switches a new bitrate value is generated until the next switch change and the change_ack signal is generated after the change lasting one period of the clock signal.

UART

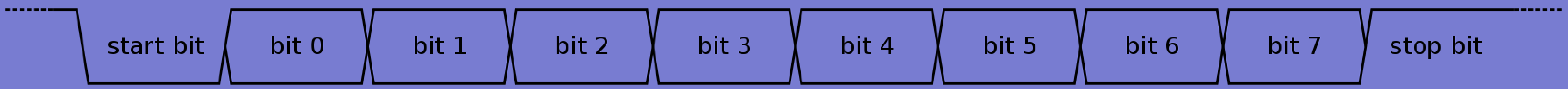
Description:

Implements a universal asynchronus receiver-transmitter.

Bit rate values:

"000"	=>	bit rate = 1200 bits/s
"001"	=>	bit rate = 2400 bits/s
"010"	=>	bit rate = 4800 bits/s
"011"	=>	bit rate = 9600 bits/s
"100"	=>	bit rate = 19200 bits/s
"101"	=>	bit rate = 38400 bits/s
"110"	=>	bit rate = 57600 bits/s
"111"	=>	bit rate = 115200 bits/s

Data: Start bit value 0, 8 data bits and stop bit value 1



UART

Inputs/Outputs:

General:

reset	- reset signal
clock	- system clock 50MHz

Bit rate:

bitrate_in	- input array that generates bit rate value

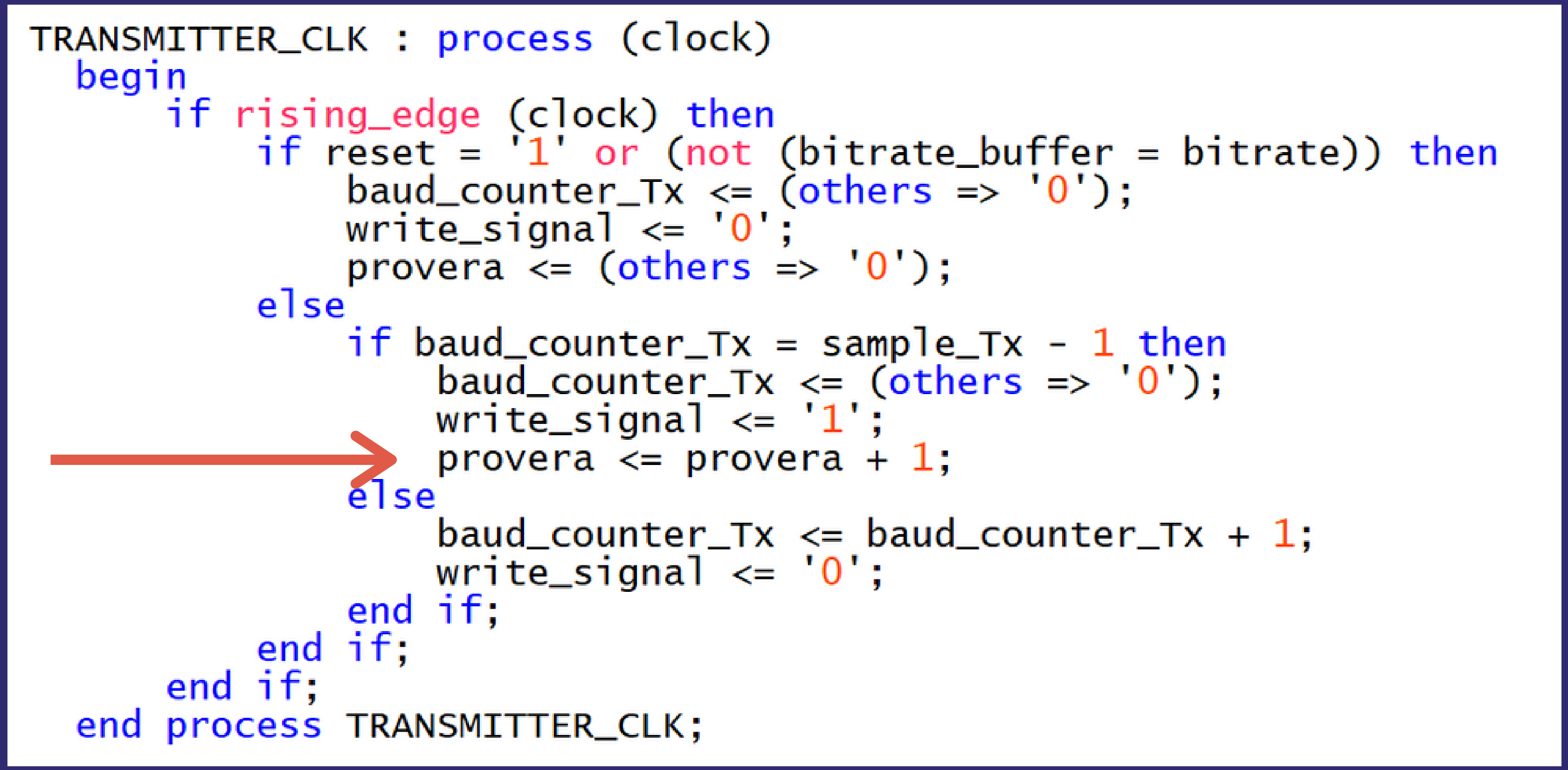
Transmitter:

data_in	- input 8 bit data to transmit
data_in_ack	- input signal that says data is in and ready to transmit
tx	- serial data output

Receiver:

data_out	- output 8bit received data
data_out_ack	- output signal that data is received
rx	- serial data input

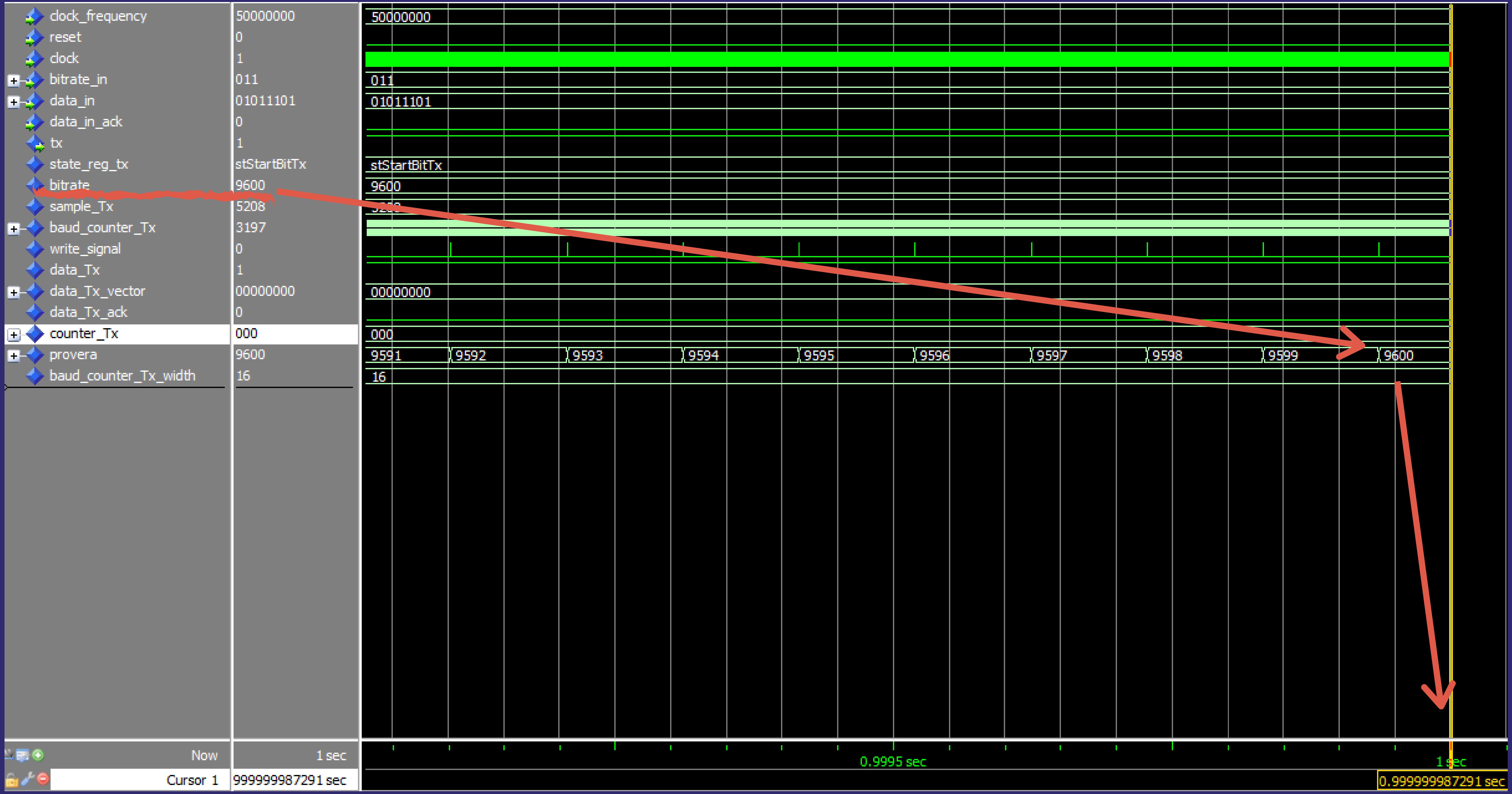
UART: bit rate



```
TRANSMITTER_CLK : process (clock)
  begin
      if rising_edge (clock) then
          if reset = '1' or (not (bitrate_buffer = bitrate)) then
              baud_counter_Tx <= (others => '0');
              write_signal <= '0';
              provera <= (others => '0');
          else
              if baud_counter_Tx = sample_Tx - 1 then
                  baud_counter_Tx <= (others => '0');
                  write_signal <= '1';
                  provera <= provera + 1;
              else
                  baud_counter_Tx <= baud_counter_Tx + 1;
                  write_signal <= '0';
              end if;
          end if;
      end if;
  end process TRANSMITTER_CLK;
```

Data is being sent to serial output on every write signal. Signal "provera" counts how many times a signal write has occured. Next images show that the number of bits sent to the serial output in one second really is same as bit rate, i.e., the condition is met.

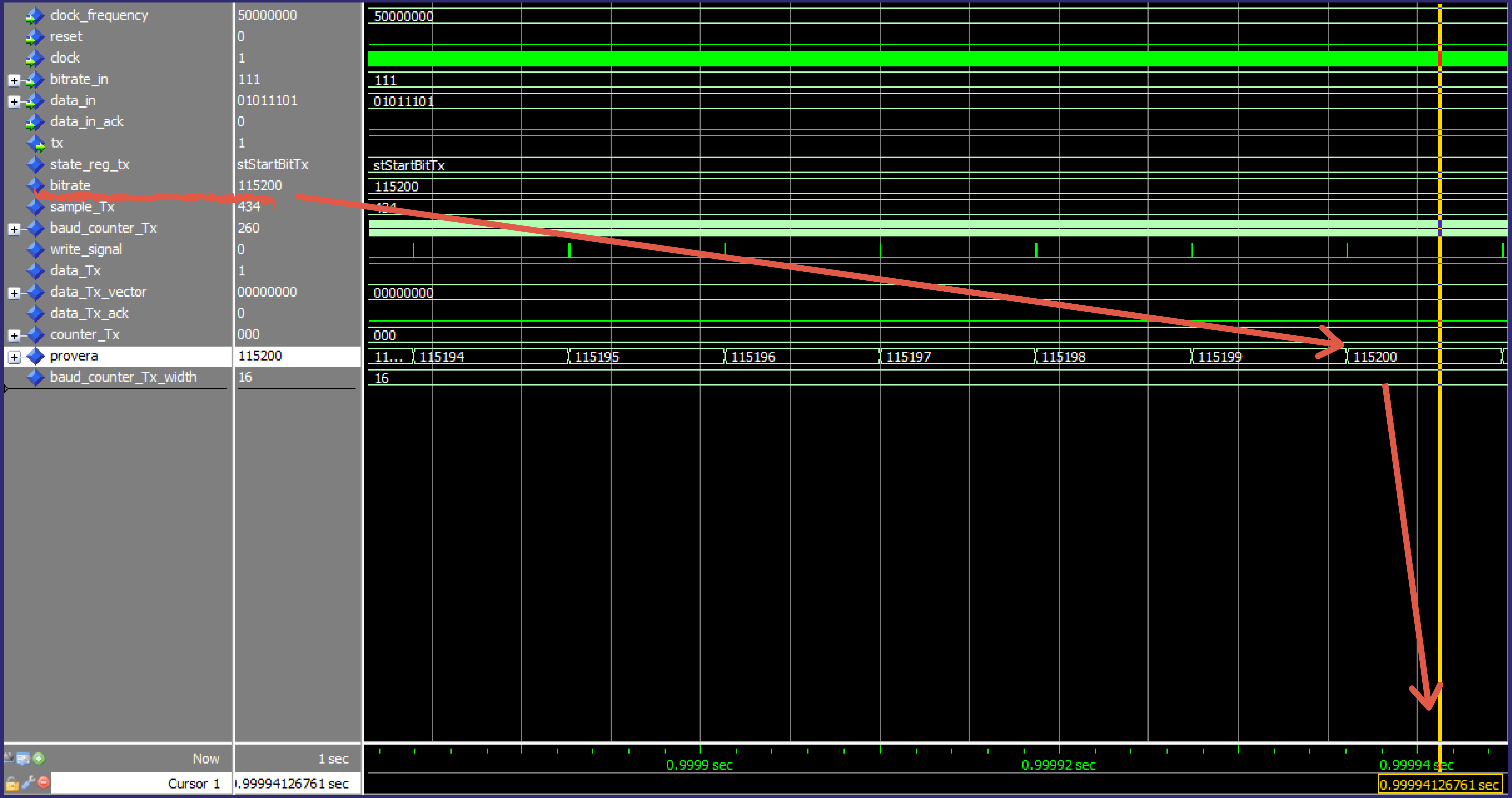
UART: bit rate



```
clock_frequency       50000000
reset                 0
clock                 1
bitrate_in            011
data_in               01011101
data_in_ack           0
tx                    1
state_reg_tx          stStartBitTx
bitrate               9600
sample_Tx             5208
baud_counter_Tx       3197
write_signal          0
data_Tx               1
data_Tx_vector        00000000
data_Tx_ack           0
counter_Tx            000
provera               9600
baud_counter_Tx_width 16

Now       1 sec
Cursor 1  999999987291 sec
```

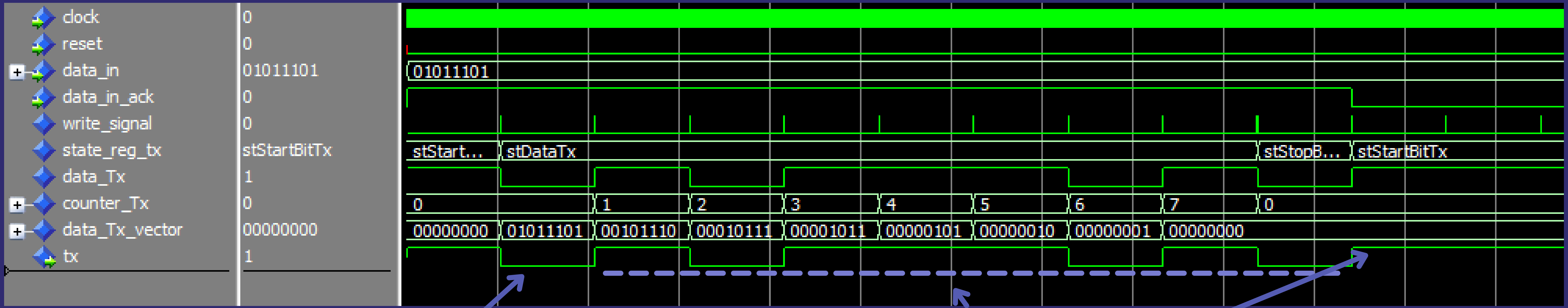
UART: bit rate



```
clock_frequency        50000000
reset                  0
clock                  1
bitrate_in             111
data_in                01011101
data_in_ack            0
tx                     1
state_reg_tx           stStartBitTx
bitrate                115200
sample_Tx              434
baud_counter_Tx        260
write_signal           0
data_Tx                1
data_Tx_vector         00000000
data_Tx_ack            0
counter_Tx             000
provera                115200
baud_counter_Tx_width  16
```

```
Now       1 sec
Cursor 1  0.99994126761 sec
```

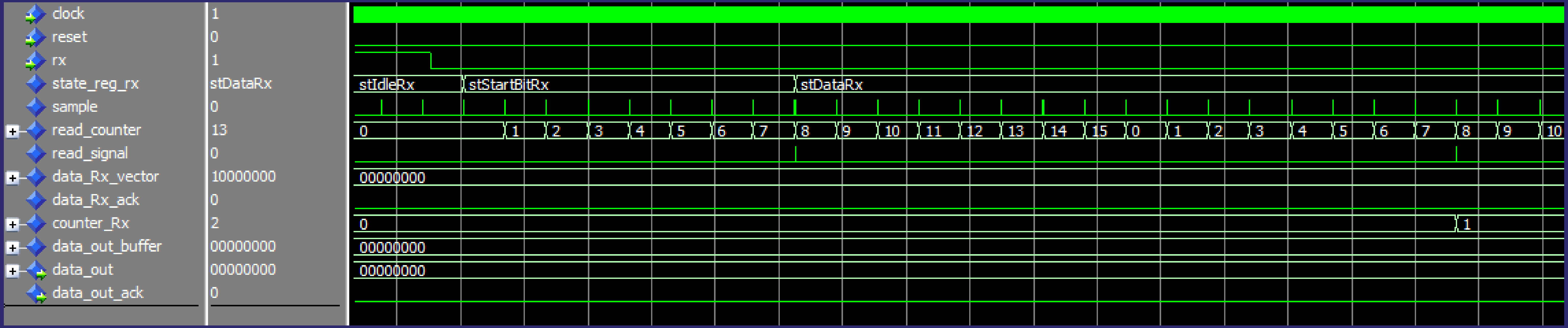
UART: transmit



```
clock           0
reset           0
data_in         01011101
data_in_ack     0
write_signal    0
state_reg_tx    stStartBitTx
data_Tx         1
counter_Tx      0
data_Tx_vector  00000000
tx              1
```

The figure shows the transmit process. When data is in (data_in_ack is high) and on the first write signal start bit '0' is sent to the serial output. After that 8 data bits are sent on write signal to the output and then stop bit '1'.

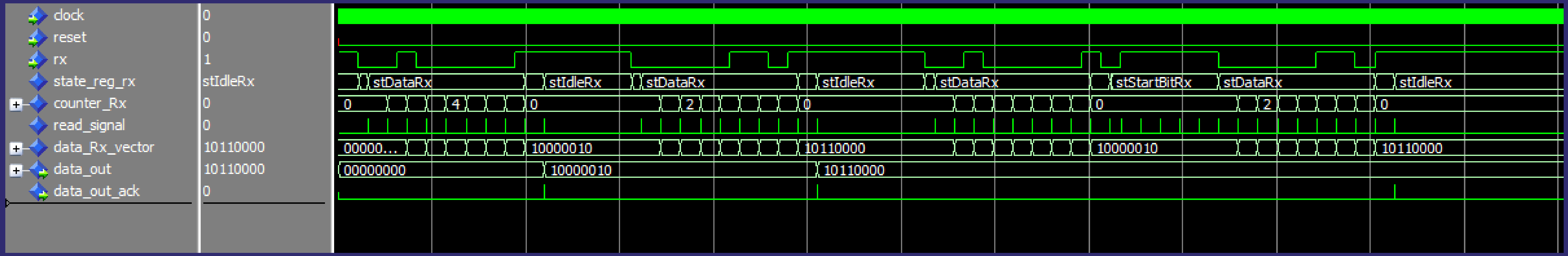
UART: receive



clock	1
reset	0
rx	1
state_reg_rx	stDataRx
sample	0
read_counter	13
read_signal	0
data_Rx_vector	10000000
data_Rx_ack	0
counter_Rx	2
data_out_buffer	00000000
data_out	00000000
data_out_ack	0

In state stIdleRx if, on sample signal, zero is detected on rx input signal, read signals are being generated on every 7th sample to be sure to read every input bit right. If on the first read signal there is still zero value that means it is start bit and machine goes to the stDataRx state where next 8 bits that represent data bits are read.

UART: receive



```
clock           0
reset           0
rx              1
state_reg_rx    stIdleRx
counter_Rx      0
read_signal     0
data_Rx_vector  10110000
data_out        10110000
data_out_ack    0
```

After 8 data bits, machine goes to the state stStopRx where it checks if stop bit is really '1'. If it is not, data is not sent to the output. On the figure it is shown that first and second data are read correctly. The third data is not read because stop bit is not correct and fourth bit is read correctly. Data_out_ack is '1' when data is read correctly and sent to the output.

LIFO MEMORY

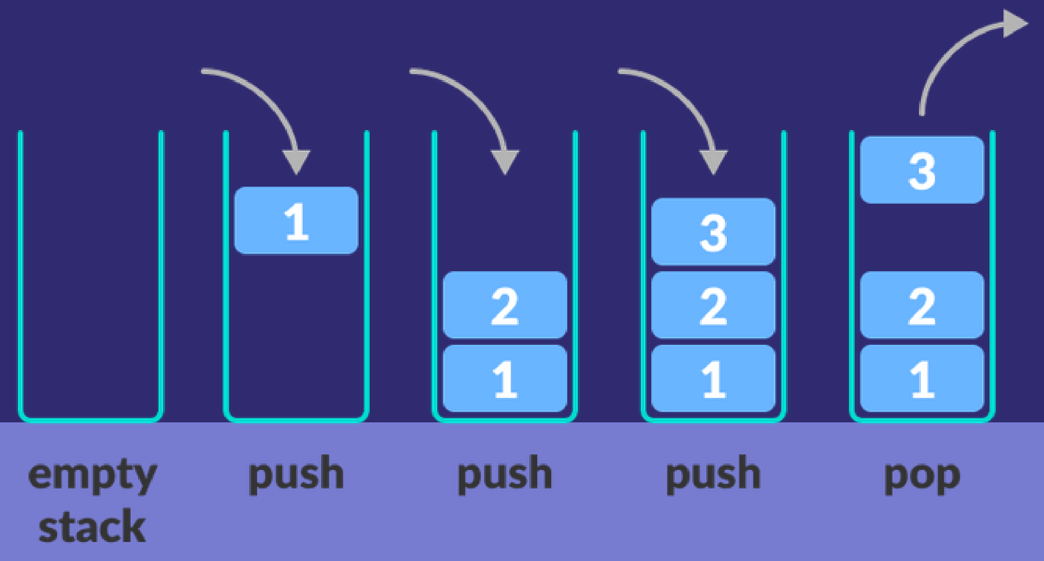
Description:

Implements a simple stack with a generic data size and a generic memory size that could be changed dpending on the need. The stack has two control signals, push and pop. When push is high, the value on the data input is added to the stack. When pop is high the top value on the stack is removed. Any attempt to add a value to a full stack or pop from empty stack is ignored and error outputs (is_full and is_empty) are generated.

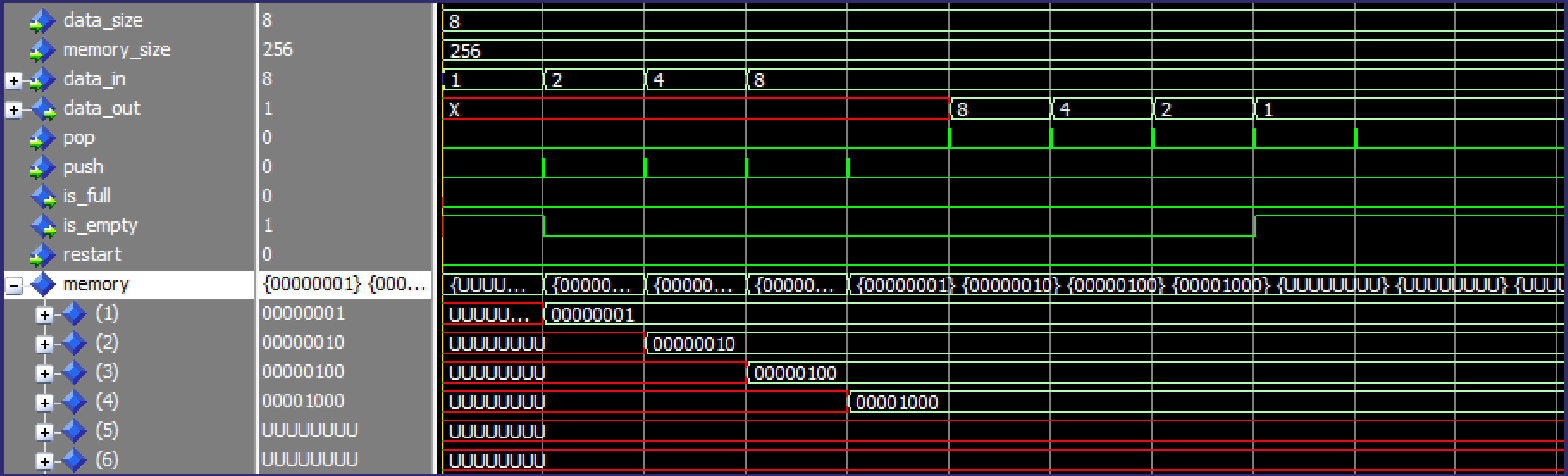
LIFO MEMORY

Inputs/Outputs:

clock	- system clock
restart	- reset signal
data_in	- input value to be put on top
data_out	- output value removed from the top
pop	- input value if high then remove data from the top
push	- input value if high then write data to the top
is_full	- output value if high that means stack is full
is_empty	- output value if high that means stack is empty



LIFO MEMORY



data_size	8
memory_size	256
data_in	8
data_out	1
pop	0
push	0
is_full	0
is_empty	1
restart	0
memory	{00000001} {000…
(1)	00000001
(2)	00000010
(3)	00000100
(4)	00001000
(5)	UUUUUUUU
(6)	UUUUUUUU

The image shows how the data is added to the memory on the push signal, and on the pop signal data is written from the memory in LIFO order.

and finally...

Last three components are used in order to make the last component UART LIFO.

UART LIFO

Description:

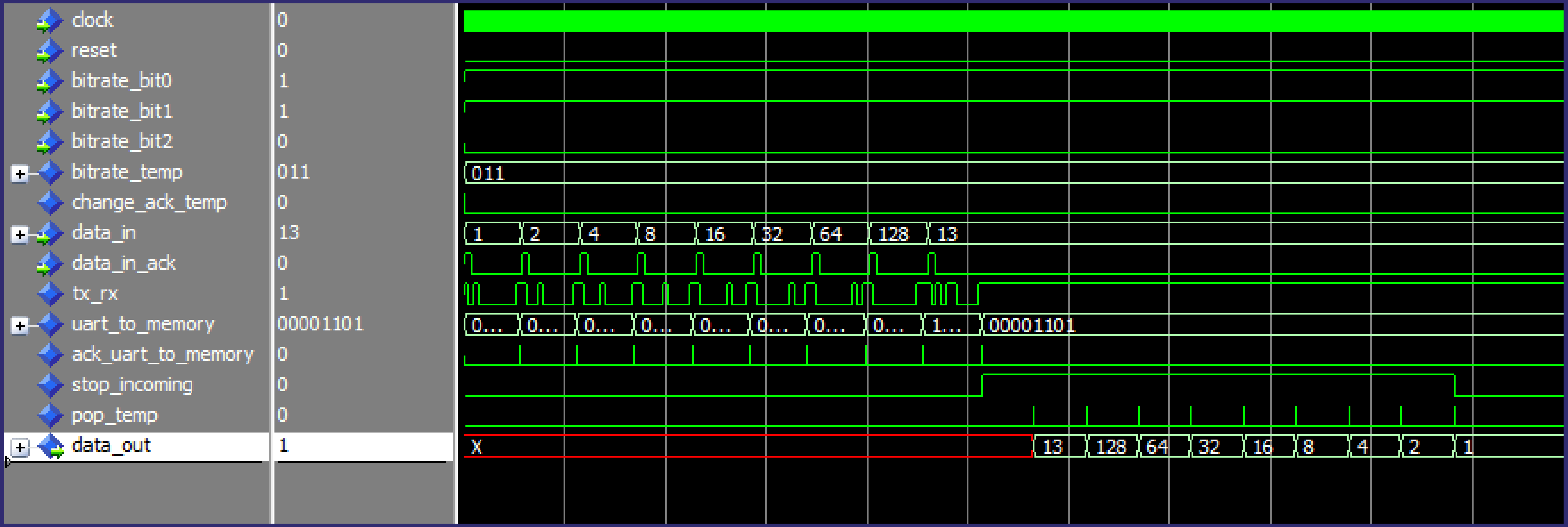
Gets input data until input value is enter (Carriage Return = "00001101"). Values go to the uart transmitter then through serial port to the uart receiver. From uart receiver output values are being stored in the LIFO memory (stack). When input value is enter, stored values are written to the output in the LIFO order.

UART LIFO

Inputs/Outputs:

clock	- system clock
restart	- reset signal
bitrate_bit0	- input values that generate bitrate value
bitrate_bit1	- -||-
bitrate_bit2	- -||-
data_in	- input data values
data_in_ack	- input signal that says data is in and ready to transmit
data_out	- output data values
error	- output signal high if memory is full

UART LIFO



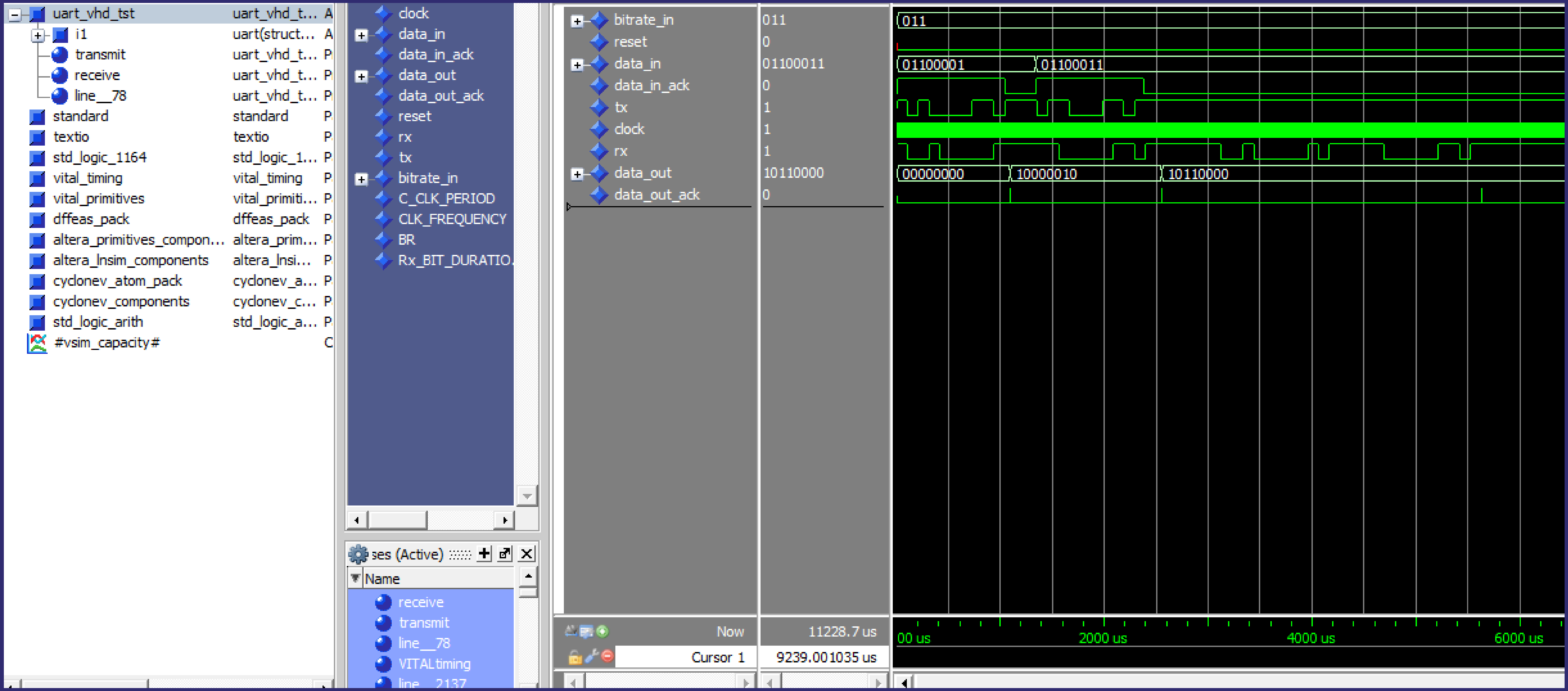
```
clock                0
reset                0
bitrate_bit0         1
bitrate_bit1         1
bitrate_bit2         0
bitrate_temp         011
change_ack_temp      0
data_in              13
data_in_ack          0
tx_rx                1
uart_to_memory       00001101
ack_uart_to_memory   0
stop_incoming        0
pop_temp             0
data_out             1
```

Input data (data_in) goes from uart transmitter through serial port (tx_rx) to uart receiver then to lifo memory (uart_to_memory). On ack_uart_to_memory data is pushed on top of the stack until the data is equal carriage return when all data written is being sent to the output (data_out) in the reverse order.

Post Synthesis

Because the UART LIFO design is too big, Quartus Prime Lite Edition does not support the simulation so Gate Level Simulation is done for UART component.

Post Synthesis: UART



```
uart_vhd_tst                uart_vhd_t...  A
  i1                        uart(struct...  A
  transmit                  uart_vhd_t...  P
  receive                   uart_vhd_t...  P
  line__78                  uart_vhd_t...  P
standard                    standard       P
textio                      textio         P
std_logic_1164              std_logic_1...  P
vital_timing                vital_timing   P
vital_primitives            vital_primiti... P
dffeas_pack                 dffeas_pack    P
altera_primitives_compon... altera_prim... P
altera_lnsim_components     altera_lnsi... P
cyclonev_atom_pack          cyclonev_a...  P
cyclonev_components         cyclonev_c...  P
std_logic_arith             std_logic_a... P
#vsim_capacity#                            C
```

```
clock
data_in
data_in_ack
data_out
data_out_ack
reset
rx
tx
bitrate_in
C_CLK_PERIOD
CLK_FREQUENCY
BR
Rx_BIT_DURATIO.
```

```
ses (Active)
Name
receive
transmit
line__78
VITALtiming
line__2137
```

```
bitrate_in     011
reset          0
data_in        01100011
data_in_ack    0
tx             1
clock          1
rx             1
data_out       10110000
data_out_ack   0

Now            11228.7 us
Cursor 1       9239.001035 us
```

further possible refinements

change #1

For data to be sent back to computer, there shoud be another UART component.

change #2

Incoming data can be put to memory (FIFO) so it doesn't depend on serial communication speed.

Author

Andrea Ćirić

andreaciric23@gmail.com

github.com/andreaciric
linkedin.com/in/andrea-ciric/