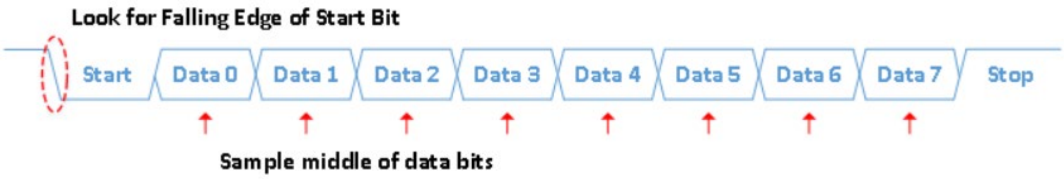
UART Transmitter

Team6

Introduction:

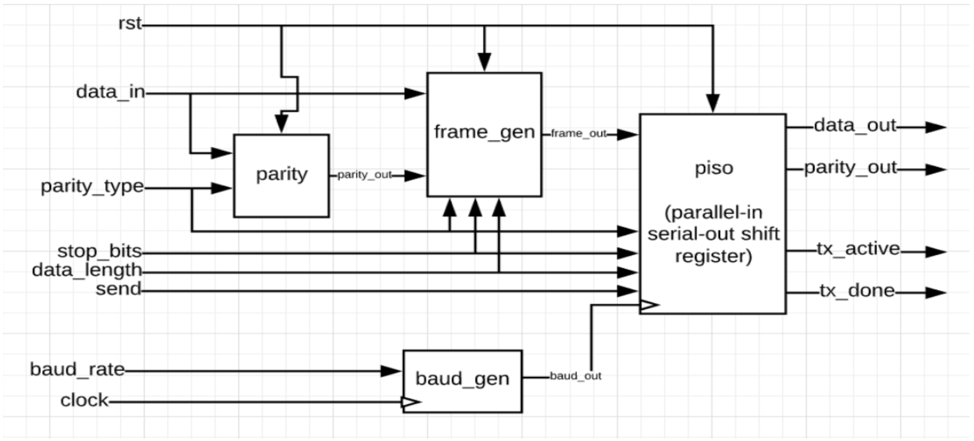
A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. Specifically, it provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices. it Converts the bytes received from the computer along parallel circuits into a single serial bit stream for outbound transmission. And then Adds a parity bit (if it's been selected) on outbound transmissions and checks the parity of incoming bytes (if selected) and discards the parity bit. In order to receive your data correctly, the transmitter and receiver must agree on the baud rate. The baud rate is the rate at which the data is transmitted. For example, 9600 baud mean 9600 bits per second.


Look for Falling Edge of Start Bit
Start
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Stop
Sample middle of data bits

The transmitter consists of 4 sub modules :

- Parity generation
- Frame generator
- Baud rate generator
- PISO register

Architecture of the transmitter



Parity generator

signal	type
data_in	Input : 8 or 7 bits
Parity_type	Input :2-bits input specifies parity type
reset	Input :Master reset for all modules
Parity out	1-bit output

After the data is received the module checks the parity type in which we have 4 cases:

- No parity :

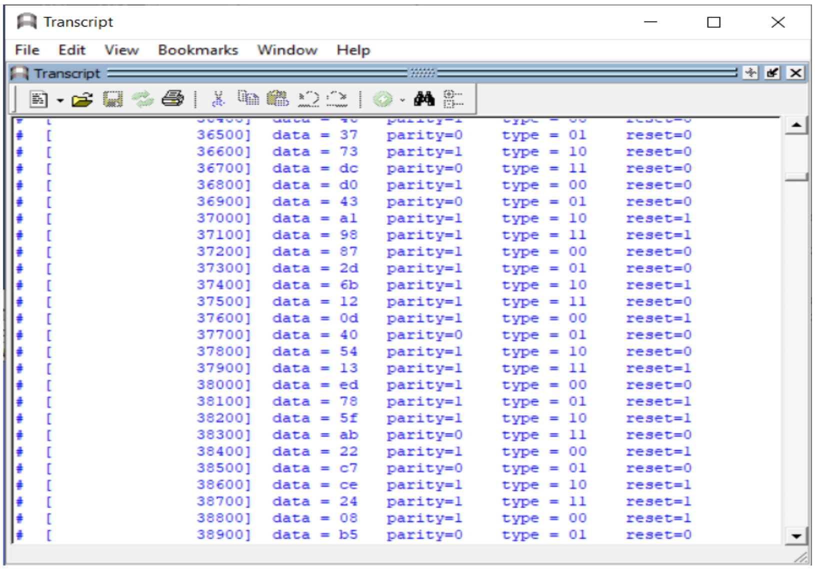
  In this case the parity outputted equal '1'
- Odd parity:

  If the number of ones in data is odd then parity ='0' other otherwise '1'
- Even parity:

  If the number of ones in data is even then parity ='0' other otherwise '1'
- Parallel parity:

  it can be considered somewhat like no parity case but in this case the parity Is odd and connected PISO register

Testbench result :



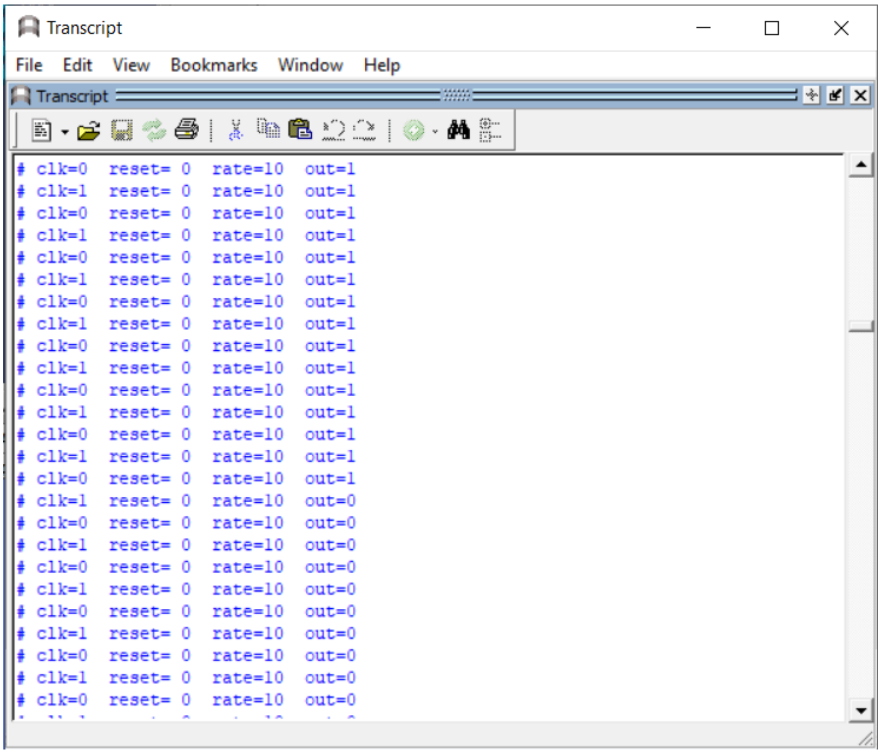
```
# [                36500]  data = 37  parity=0   type = 01   reset=0
# [                36600]  data = 73  parity=1   type = 10   reset=0
# [                36700]  data = dc  parity=0   type = 11   reset=0
# [                36800]  data = d0  parity=1   type = 00   reset=0
# [                36900]  data = 43  parity=0   type = 01   reset=0
# [                37000]  data = a1  parity=1   type = 10   reset=1
# [                37100]  data = 98  parity=1   type = 11   reset=1
# [                37200]  data = 87  parity=1   type = 00   reset=0
# [                37300]  data = 2d  parity=1   type = 01   reset=0
# [                37400]  data = 6b  parity=1   type = 10   reset=1
# [                37500]  data = 12  parity=1   type = 11   reset=0
# [                37600]  data = 0d  parity=1   type = 00   reset=1
# [                37700]  data = 40  parity=0   type = 01   reset=0
# [                37800]  data = 54  parity=1   type = 10   reset=0
# [                37900]  data = 13  parity=1   type = 11   reset=1
# [                38000]  data = ed  parity=1   type = 00   reset=0
# [                38100]  data = 78  parity=1   type = 01   reset=1
# [                38200]  data = 5f  parity=1   type = 10   reset=1
# [                38300]  data = ab  parity=0   type = 11   reset=0
# [                38400]  data = 22  parity=1   type = 00   reset=1
# [                38500]  data = c7  parity=0   type = 01   reset=0
# [                38600]  data = ce  parity=1   type = 10   reset=1
# [                38700]  data = 24  parity=1   type = 11   reset=1
# [                38800]  data = 08  parity=1   type = 00   reset=1
# [                38900]  data = b5  parity=0   type = 01   reset=0
```

Baud rate generator:

signal	type
clock	Input:50MHz frequency
baud rate	2-bits input specify baud rate
reset	Master reset for all modules
Baud_out	1-bit output in the form of pulses

we can control frequency of input clock through it, we have four choices for our project each choice make baud rate works as a clock with specific frequency for example, if user selects "00" baud rate will change its state from 0 to 1 or the opposite every 1063 cycles of input clock and so on for other cases integer 'I' changes when user enter his select and has a value relative to user select we have a counter which increments by 1 when counter changes its value we check a statement using always block if this statement is valid the baud rate will change output if not the output will be the same the statement is (count % i) so at 'I' or its multiples the output will change

Testbench result :



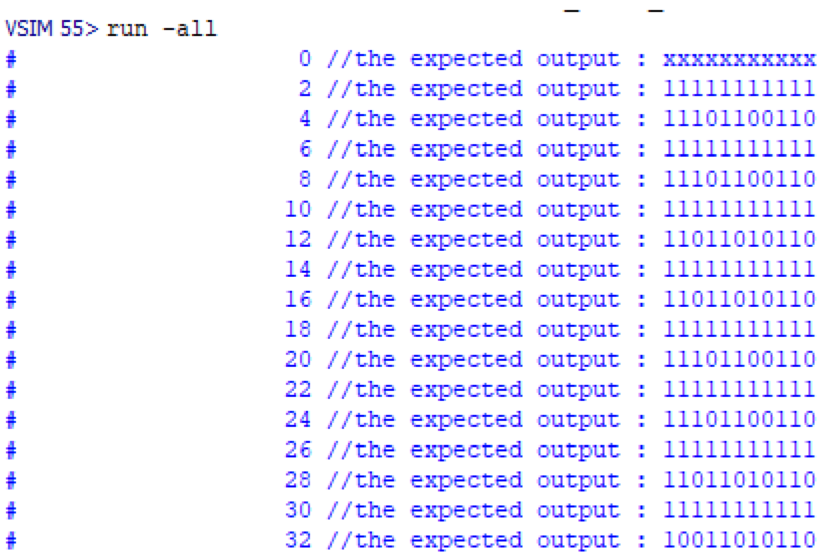
```
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=1
# clk=0  reset= 0  rate=10  out=1
# clk=1  reset= 0  rate=10  out=0
# clk=0  reset= 0  rate=10  out=0
# clk=1  reset= 0  rate=10  out=0
# clk=0  reset= 0  rate=10  out=0
# clk=1  reset= 0  rate=10  out=0
# clk=0  reset= 0  rate=10  out=0
# clk=1  reset= 0  rate=10  out=0
# clk=0  reset= 0  rate=10  out=0
# clk=1  reset= 0  rate=10  out=0
# clk=0  reset= 0  rate=10  out=0
```

Frame generator :

signal	type
Parity_out	Input: 1-bit the output of parity module
baud rate	Input:2-bits input specifies baud rate
reset	Input: Master reset for all modules
stop bits	Input: low when using 1 stop bit, high when using two stop bits
data_length	Input : 1-bit low when using 7 data bits, high when using 8
Data_in	Input: 8 or 7 bits
Parity_type	Input :2-bits input specifies parity type
Frame_out	Output : 12-bit contain the data_in , stop bits, start bits and parity

It receives the input data ,getting its size form data length, and concatenates the start bit at first . then check the if there is no parity or not . finally, it adds the stop bits based on the input then sends the output to the PISO register

Testbench result:



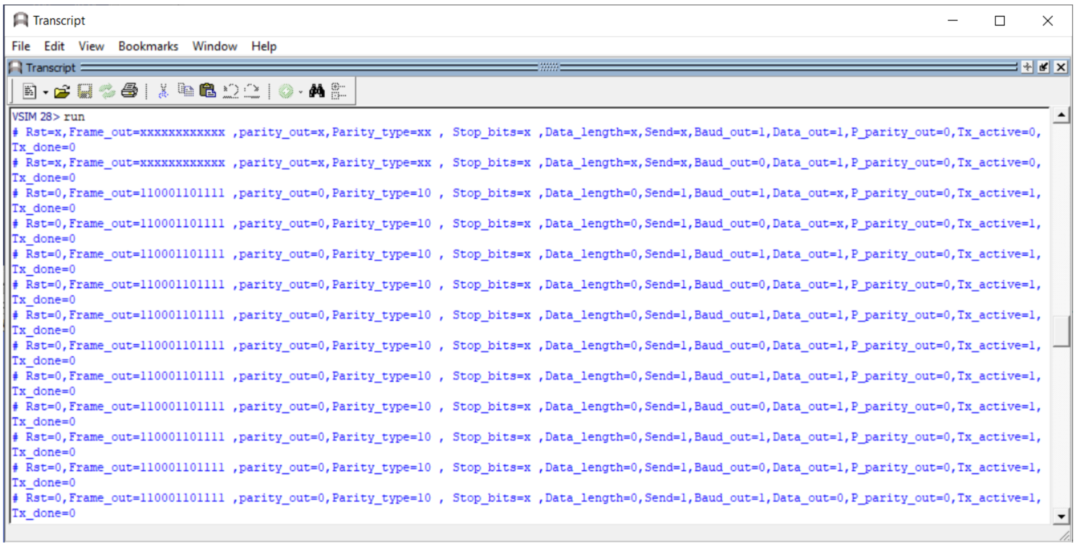
```
VSIM 55> run -all
#                    0 //the expected output : xxxxxxxxxxx
#                    2 //the expected output : 11111111111
#                    4 //the expected output : 11101100110
#                    6 //the expected output : 11111111111
#                    8 //the expected output : 11101100110
#                   10 //the expected output : 11111111111
#                   12 //the expected output : 11011010110
#                   14 //the expected output : 11111111111
#                   16 //the expected output : 11011010110
#                   18 //the expected output : 11111111111
#                   20 //the expected output : 11101100110
#                   22 //the expected output : 11111111111
#                   24 //the expected output : 11101100110
#                   26 //the expected output : 11111111111
#                   28 //the expected output : 11011010110
#                   30 //the expected output : 11111111111
#                   32 //the expected output : 10011010110
```

PISO register:

signal	type
Frame_out	Input: 12-bit contain the data_in , stop bits, start bits and parity
Baud_out	Input: 1-bit in the form of pulses
Data_in	Input : 8 or 7 bits
reset	Input: Master reset for all modules
stop bits	Input: low when using 1 stop bit, high when using two stop bits
data_length	Input : 1-bit low when using 7 data bits, high when using 8
send	Input: 1-bit high when sending data otherwise low
Parity_type	Input :2-bits input specifies parity type
Data_out	Output : output data sent to the receiver
P_parity	Output : 1-bit parallel odd parity output, low when using the frame parity.
Tx_active	Output:1-bit high when transmitting otherwise low
Tx_done	Output: 1-bit high when transmitting is done

receive all the outputs from previous circuits and then outputs it Serial. By shifting all the input

Testbench result:

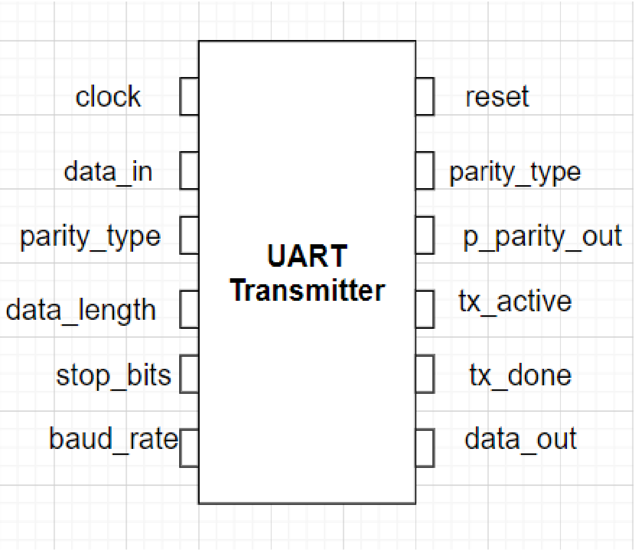


```
VSIM 28> run
# Rst=x,Frame_out=xxxxxxxxxxxx ,parity_out=x,Parity_type=xx , Stop_bits=x ,Data_length=x,Send=x,Baud_out=1,Data_out=1,P_parity_out=0,Tx_active=0,
Tx_done=0
# Rst=x,Frame_out=xxxxxxxxxxxx ,parity_out=x,Parity_type=xx , Stop_bits=x ,Data_length=x,Send=x,Baud_out=0,Data_out=1,P_parity_out=0,Tx_active=0,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=x,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=0,Data_out=x,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=0,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=0,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=0,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=0,Data_out=1,P_parity_out=0,Tx_active=1,
Tx_done=0
# Rst=0,Frame_out=110001101111 ,parity_out=0,Parity_type=10 , Stop_bits=x ,Data_length=0,Send=1,Baud_out=1,Data_out=0,P_parity_out=0,Tx_active=1,
Tx_done=0
```

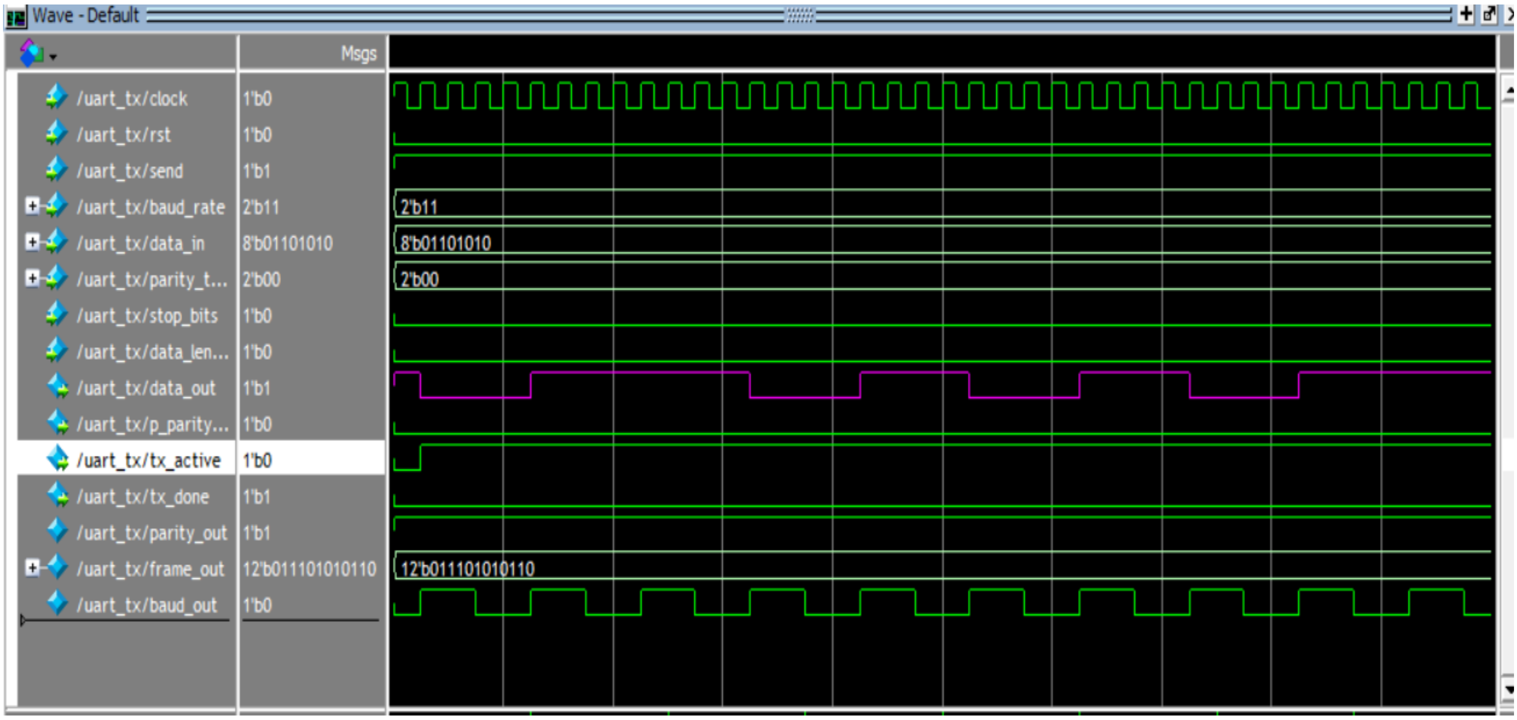
Top module:

signal	type
clock	Input: 50MHz frequency
baud rate	Input:2-bits input specifies baud rate
Data_in	Input : 8 or 7 bits
reset	Input: Master reset for all modules
stop bits	Input: low when using 1 stop bit, high when using two stop bits
data_length	Input : 1-bit low when using 7 data bits, high when using 8
send	Input: 1-bit high when sending data otherwise low
Parity_type	Input :2-bits input specifies parity type
Data_out	Output : output data sent to the reciever
P_parity	Output : 1-bit parallel odd parity output, low when using the frame parity.
Tx_active	Output:1-bit high when transmitting otherwise low
Tx_done	Output: 1-bit high when transmitting is done

IC pin diagram:

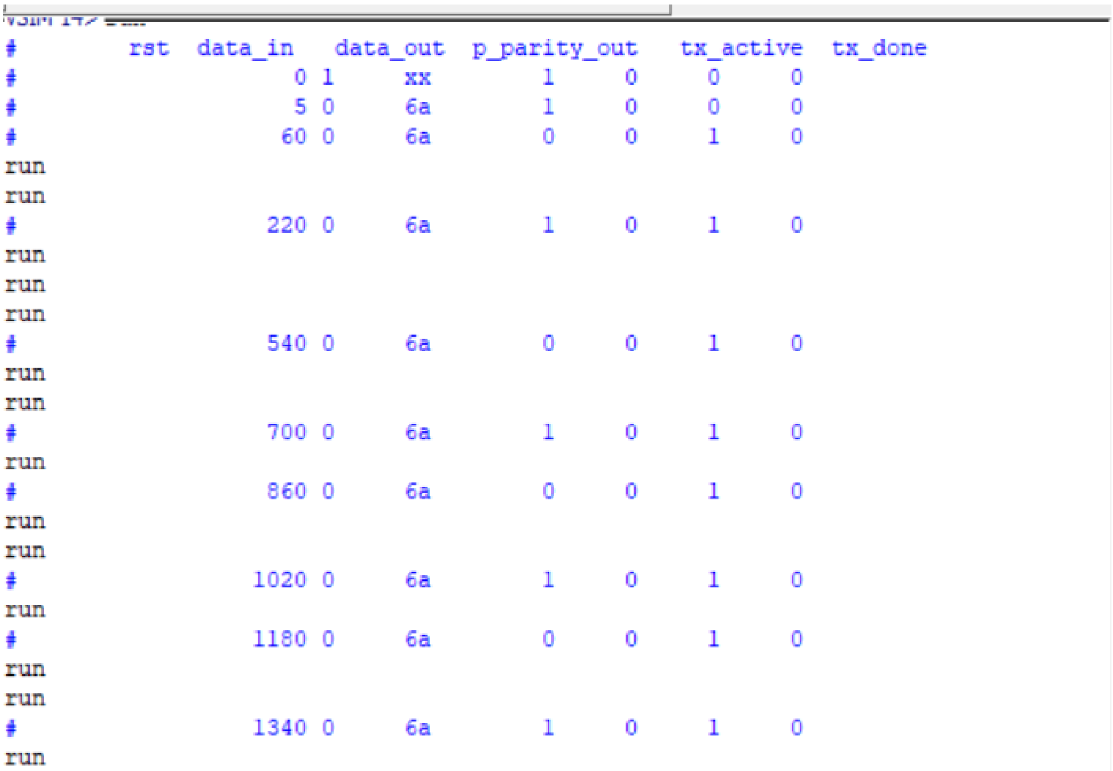


Wave simulation output:



	Msgs
/uart_tx/clock	1'b0
/uart_tx/rst	1'b0
/uart_tx/send	1'b1
/uart_tx/baud_rate	2'b11
/uart_tx/data_in	8'b01101010
/uart_tx/parity_t...	2'b00
/uart_tx/stop_bits	1'b0
/uart_tx/data_len...	1'b0
/uart_tx/data_out	1'b1
/uart_tx/p_parity...	1'b0
/uart_tx/tx_active	1'b0
/uart_tx/tx_done	1'b1
/uart_tx/parity_out	1'b1
/uart_tx/frame_out	12'b011101010110
/uart_tx/baud_out	1'b0

Testbench results:



```
VSIM 14> run
#        rst  data_in   data_out  p_parity_out    tx_active  tx_done
#                  0  1       xx         1      0      0      0
#                  5  0       6a         1      0      0      0
#                 60  0       6a         0      0      1      0
run
run
#                220  0       6a         1      0      1      0
run
run
run
#                540  0       6a         0      0      1      0
run
run
#                700  0       6a         1      0      1      0
run
#                860  0       6a         0      0      1      0
run
run
#               1020  0       6a         1      0      1      0
run
#               1180  0       6a         0      0      1      0
run
run
#               1340  0       6a         1      0      1      0
run
```