1- The following code solves the problem.

```
#include "uart_transmitter.h"

typedef enum{idle, transmit} uart_transmit_states_type;

void uart_transmitter(bool &uart_tx, ap_uint<8> data, bool
baud_rate_signal, bool start) {
#pragma HLS INTERFACE ap_none port=data
#pragma HLS INTERFACE ap_none port=baud_rate_signal
#pragma HLS INTERFACE ap_none port=uart_tx
#pragma HLS INTERFACE ap_none port=start
#pragma HLS INTERFACE ap_ctrl_none port=return


	bool even_parity = data.xor_reduce();
	ap_uint<12> d = ((ap_int<2>)1, (bool)even_parity, (ap_int<8>)data,
(bool)0b0);


	static unsigned int 		bit_counter = 0;
	static uart_transmit_states_type state = idle;


	uart_transmit_states_type next_state;
	unsigned int 		next_bit_counter;
	bool uart_tx_local;


	switch(state) {
	case idle:
		if ( start == 1) {
			next_state = transmit;
			uart_tx_local	= 1;
			next_bit_counter	= 0;
		} else {
			next_state = idle;
			uart_tx_local	= 1;
			next_bit_counter	= 0;
		}

		break;
	case transmit:
		if (baud_rate_signal == 1) {
			if (bit_counter == 12) {
				next_state		= idle;
				uart_tx_local	= 1;
				next_bit_counter	= 0;
			} else  {
				next_state	=  transmit;
				uart_tx_local	=  d[bit_counter];
				next_bit_counter =  bit_counter+1;
			}
		} else {
			if (bit_counter == 0) {
```

```
                        uart_tx_local    =  1;
                    } else {
                        uart_tx_local    =  d[bit_counter-1];
                    }
                    next_state       =  transmit;
                    next_bit_counter =  bit_counter;
                }

                break;

        default:
                break;
        }


        state                   = next_state;
        bit_counter             = next_bit_counter;
        uart_tx                 = uart_tx_local;

}
```

2- To cope with this problem, we should modify the baud rate generator and the UART receiver IPs. The following codes show these modifications.

```
//9600 baud rate --> BAUD_RATE_NUMBER = (1s/14400)/10ns =
(1000000000/14400)/10 = 6944
#define BAUD_RATE_NUMBER 6944

//for simulation
//#define BAUD_RATE_NUMBER 20

typedef enum{zero, one} baud_rate_states_type;
void baud_rate_generator(bool &baud_rate_signal) {
#pragma HLS INTERFACE ap_none port=baud_rate_signal
#pragma HLS INTERFACE ap_ctrl_none port=return


        static baud_rate_states_type state = zero;
        static unsigned int counter = BAUD_RATE_NUMBER-1;


        baud_rate_states_type next_state;
        unsigned int          next_counter;

        bool baud_rate_signal_local;

        switch(state) {
        case zero:
                if (counter == 1) {
                        next_counter           = counter-1;
                        next_state             = one;
```

```
            } else {
                  next_counter           = counter-1;
                  next_state             = zero;
            }
            baud_rate_signal_local = 0;
            break;
      case one:
            next_counter           = BAUD_RATE_NUMBER-1;
            next_state             = zero;
            baud_rate_signal_local = 1;

            break;
      default:
            break;
      }

      state = next_state;
      counter = next_counter;
      baud_rate_signal = baud_rate_signal_local;
}
```

```
typedef enum{idle, receive} uart_receive_states_type;

void uart_receiver (bool uart_rx, bool baud_rate_signal, ap_uint<8> &data, bool
&valid_data) {
#pragma HLS INTERFACE ap_none port=valid_data
#pragma HLS INTERFACE ap_none port=data
#pragma HLS INTERFACE ap_none port=baud_rate_signal
#pragma HLS INTERFACE ap_none port=uart_rx
#pragma HLS INTERFACE ap_ctrl_none port=return


      static ap_uint<8> d;
      static unsigned int bit_counter = 0;
      static uart_receive_states_type state = idle;


      uart_receive_states_type next_state;
      unsigned int next_bit_counter ;

      bool stop_bit;
      bool odd_parity_bit;
      bool valid_data_local = 0;



      switch(state) {
      case idle:
            if (baud_rate_signal == 1) {
                  if (uart_rx == 0) {
                        next_state =  receive;
```

```
                } else {
                        next_state = idle;
                }
        } else {
                next_state = idle;
        }
        valid_data_local = 0;
        next_bit_counter = 0;
        break;
case receive:
        if (baud_rate_signal == 1) {
                if (bit_counter == 9) {
                        stop_bit = uart_rx;
                        if (stop_bit == 1) {
                                valid_data_local = 1;
                        } else {
                                valid_data_local = 0;
                        }
                        next_bit_counter = bit_counter+1;
                        next_state = idle;
                } else if (bit_counter == 8) {
                        odd_parity_bit = uart_rx;
                        if (odd_parity_bit == !d.xor_reduce()) {
                                valid_data_local = 1;
                        } else {
                                valid_data_local = 0;
                        }
                        next_bit_counter = bit_counter+1;
                        next_state = idle;
                } else {
                        d[bit_counter] = uart_rx;
                        next_bit_counter = bit_counter+1;
                        next_state = state;
                        valid_data_local = 0;
                }
        } else {
                valid_data_local = 0;
                next_state = receive;
                next_bit_counter = bit_counter;
        }
        break;
default:
        break;

}

state = next_state;
bit_counter = next_bit_counter;
data = d;
valid_data = valid_data_local;
}
```