Asynchronous Serial Receiver Report

Geoff Watson & Waseh Ahmad
ECE 491 Lab04 Asynchronous Serial Receiver
10-02-2017

ABSTRACT

The following report identifies an implementation of the Asynchronous Serial Receiver using a Nexyx4DDR FPGA, built with System Verilog code. This report outlines the design considerations taken into account and present a test plan consisting of requirements, tests to be carried out, and simulation outputs confirming the PASSing of those tests.

INTRODUCTION

In this lab, our goal was to implement an asynchronous serial receiver that can operate at different baud rates and receive a single or multiple bytes. In receiving this data, we must be able to identify when proper synchronization did not occur, producing a framing error, and be able to ignore noise when trying to identify a start bit, producing a spurious start. In addition, when we receive good data, the ready signal indicates to the receiver that the data is able to be read.

A detailed description of how the design requirements were met is provided in the test plan and the following simulation output.

Alternate Designs

1. The extensive use of modulation was considered in our first design. We originally had thought of separating the modules for datapath and controller where the FSM would just process use status bits sent from the datapath module and guide it to its next output. This was to be done in separate modules to increase the level of hidden design. Also, the counters used for sampling the data would be external to both the datapath and the controller.

2. Another approach that was considered involved collecting all the data i.e. at a rate of 16*BAUD. The data for each bit would then be checked for consistency, i.e. if a data bit = 0 was received, to confirm that it was 0, that data bit period would be sampled 16 times, each time confirming that a 0 was being received. This would potentially add more accuracy to the receiver, where noise could potentially modify data being sent, verifying that data received was clean.

Reason for choosing the final design

The largest factor leading to us choosing the design that we did was wanting to minimize adding too much complexity and modulation where it was not necessary. This meant keeping the counters within the FSM for easier management. We would not need separate modules, just logic variables to keep track of counting. This allowed for easier synchronization and identification of the incoming bytes. Also, since the requirements specified that we only needed to sample each assumed data bit once, we felt it would be overkill to sample it multiple times. Also, because the serial receiver might lose synchronization if a series of 0s or 1s are sent, there would not be much use in trying to improve data validation by over sampling. Our design also allows to easily change the sample frequency, which might be helpful for the Manchester receiver.

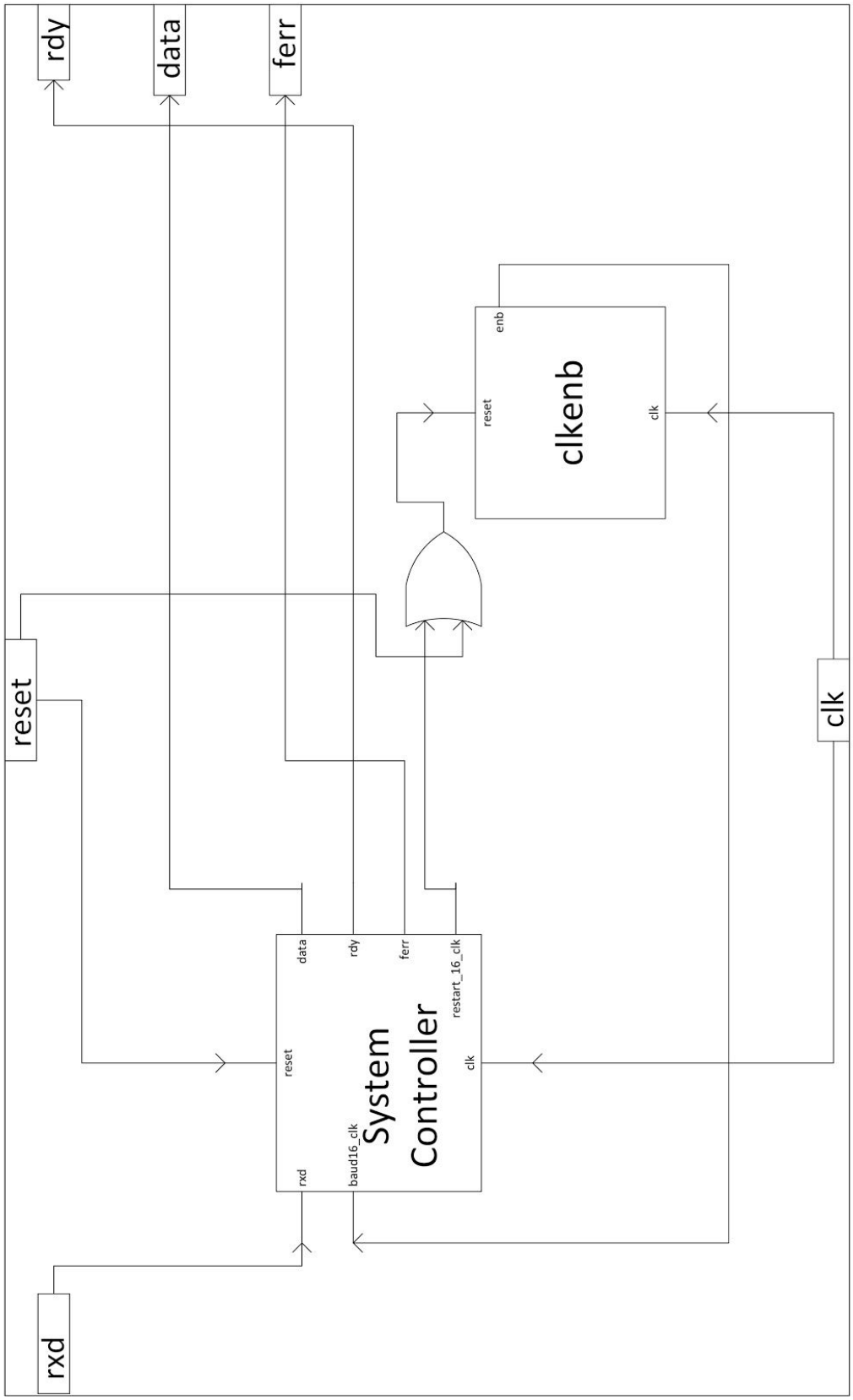
Chosen Design

The design we chose is shown below in the block diagram. The FSM is shown in the state diagram below. The receiver module contains two major modules. The **clkenb** module is responsible for providing the signal to sample the received data. This module is reset using either the master reset or a reset signal provided by the FSM (to ensure synchronicity with the remaining blocks). The **FSM** contains the majority of the functionality of the receiver. It takes in as inputs, the data signal and the sample rate clock output. The description of the FSM is provided below. It also contains internal logic variables used for counting purposes. These counters are controlled by the states and conditions within the FSM.

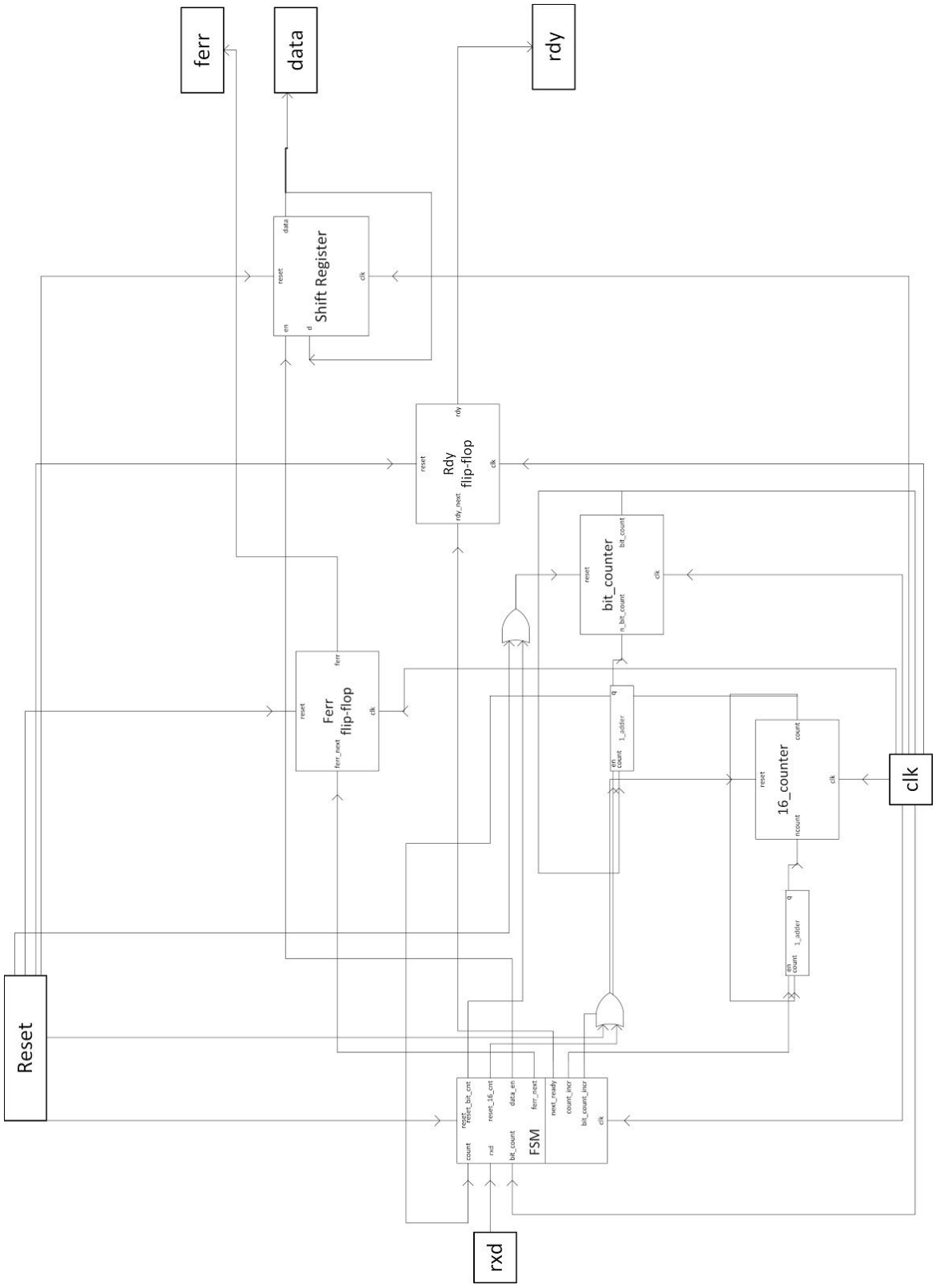
The way the receiver module design behaves is:

1. Waits for data i.e. falling edge indicating a start bit
2. Checks for spurious start by sampling the data halfway through the start bit. If a spurious start is confirmed, return to step 1.
3. Once a valid start bit is confirmed, the data signal is sampled at the middle of each assumed data bit i.e. one BAUD RATE period separations between samples.
4. As the data is sampled, it is recorded and placed in the appropriate bit-place in the output **data** signal.
5. Once the last bit is recorded, the module attempts to sample the stop bit. If the data is high at the time of the stop bit sample, **rdy** is asserted, telling the user that data is now available to be used.
6. If the sampled bit value is instead logic low, a framing error has occurred (Note that this method of checking for framing error might be erroneous if the last bit received was a logic high and the receiver did not function properly). the **ferr** output is then asserted, and stays asserted until the start of the next byte.
7. Return to step 1.

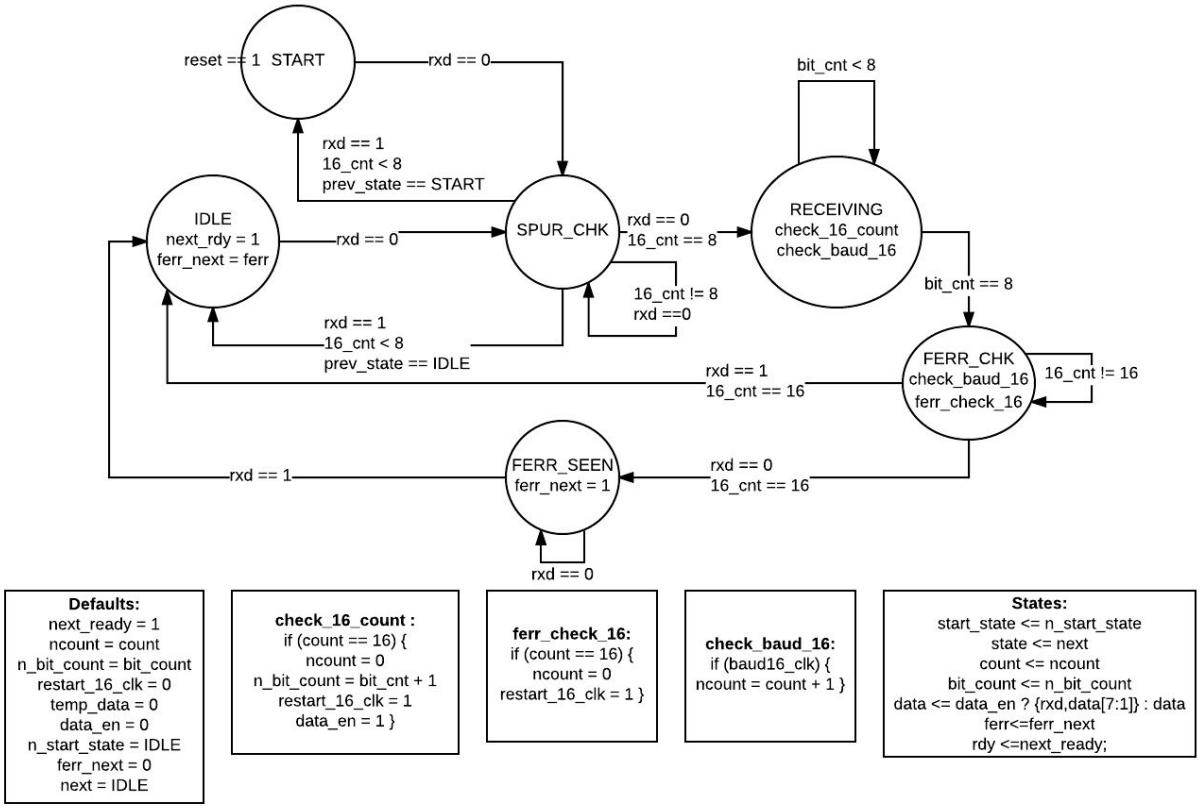
BLOCK DIAGRAM of RECEIVER



BLOCK DIAGRAM OF SYSTEM CONTROLLER MODULE



FSM design for receiver module



Above is the block diagram for our FSM. When the system is initially started, the FSM begins in the START state. We stay in the **START** state until we see that rxd goes low, indicating a possible start bit. Once rxd goes low, we enter the spurious start check state (**SPUR_CHK**). In that state, we wait for 8 ticks of the 1/16 Baud rate clock. If rxd is still low at this point, then we can say that we have seen a valid start bit, restart the 1/16 baud counter, and enter the **RECEIVING** state. If on the other hand, rxd was high, then we have seen a spurious start, and we go back to the **IDLE** state and wait for rxd to go low again. Once we are in the receiving state, we start by waiting 16 counts of the 1/16 Baud clock to get to the middle of the bit, then set data at that bit to the value of rxd. We do this 8 times to get the full byte, then enter ferr check (**FERR_CHK**) state. When we are in this state, we are making sure that there isn't a framing error and that we are seeing the stop bit. We do this by waiting 16 ticks of the 1/16 baud clock, then if rxd is high, we have seen a valid stop bit so we assert rdy to high, and enter the idle state. If we rxd is low, then we have a framing error and enter the ferr seen state(**FERR_SEEN**), where ferr is asserted and kept high. We then stay in that state until rxd goes high. THe next state is the IDLE state, where rdy is high. If ferr is also high, the user will know that although data is ready, a framing error may have occurred, corrupting the data.

Test Plan

Requirements Checklist (TEST PLAN at 5th Description)

Description	**Test Method**	**Detailed Results**
1. Module Interface	Code Inspection	All .sv files contain comments about the design. in the header portions. Comments included to assist with understanding of code. FSM and combinational logic meets standards.
2. Module function:	Demonstration in hardware using Nexys 4 DDR board using realterm and simulation. Connect with serial transmitter to show output of receiver on oscilloscope	Data transmitted via real term shows on 7 segment display with no flickering Multiple bytes show up as earliest transmitted on the furthest left of the display board. Data on oscilloscope verified to match that sent from the board. Rdy is shown to go high at the correct stages. Hardware demonstration verified by Professor Nadovich.
3. Uses Nexys4 board 100Mhz clock; all flip-flop clock inputs tied directly to this signal	- Check every flip-flop and verify that it is driven by the board 100Mhz clock	All clocks are connected to system 100 MHz clock as evaluated in **receiver_top** and **receiver_fsm** files
4. Contains no latches	Inspection of Synthesis Report	Synthesis report contains no latches reference (all outputs are defined in receiver_fsm.sv, lines 72-81)

5. Test circuit – show test that test circuit functions properly by interfacing with realterm and also using simulations The following **requirements** must be met: 1. Data reception at the BAUD rate 2. When reset is pressed, rdy is low, ferr is low 3. Spurious start shall not be acknowledged as a start 4. rdy goes high after a valid stop bit is recognized 5. rdy goes low when a valid start bit is recognized 6. ferr is asserted when a framing error is detected 7. ferr stays high until next start bit 8. Spurious start with framing error does should not change any of the outputs 9. Multiple bytes are correctly transmitted 10. A single byte is correctly transmitted	Format → Test--object being tested-- how to check for Pass **Hardware** **1) Requirement 2** a) Press the reset button b) System reset c) **rdy** and **ferr** LED are low, **data** = all zeroes **2) Requirement 10** a) Send one byte from realterm b) Correct Reception of one byte c) byte displays on 7-seg display in hex(least sig. positions), **rdy** goes high after byte, **ferr** is LOW **3) Requirement 9** a) Send multiple bytes from real term b) Correct reception of multiple bytes c) all bytes are displayed, with the most recent one, being shifted left on the 7-seg display after each new byte, **rdy** flickers as each byte is sent and then stays on, **ferr** is low throughout **4) Requirement 9** a) Send a long string of ascii characters from real term b) Timing constraints met and correct transmission c) Last four ascii values displayed in hex, **rdy** is high at the end, **ferr** never turns on 5) **Requirement 6 & 7** a) Assert break for 500ms on realterm b) Testing framing error c) **rdy** goes low for 500 ms, **ferr** is high for all 500 ms, then **rdy** goes high again after 500 ms. **ferr** remains	All tests PASS using hardware and simulation as shown below. BAUD RATE ~ 9600 (default) Hardware tests verified PASS by Professor Nadovich

	high. Data is shown to be all zeros 6) **Requirement 7** a) Send data after framing error b) testing framing error signal c) **ferr** goes low when send is pressed and then stays low, **rdy** goes high again at end of reception, new data is shown on display 7) **Requirement 3 & 8** a) Increase the baud rate on real term and try sending signals b) Spurious start check c) **Data** will not change, **rdy** will not change **ferr** will not change **Simulation** 1) **Requirement 2** a) Assert reset b) initial states c) **rdy** is low, **data** = 8'hff, **ferr** is low 2) **Requirement 3** a) Send spurious start after restart, before any data is sent b) initial conditions do not change c) **rdy** is low, **data** = 8'hff, **ferr** is low. 3) **Requirement 6** a) Assert received signal low for a time greater than a byte period. Then let it go high again b) framing error c) **ferr** should go high after a byte time period is passed, and remain high after. **rdy** stays low, **data** = 8'h00 4) **Requirement 4, 5, 9**	

	a) Send multiple bytes of data consecutively b) rdy is high for the correct period of time between bytes **c)** **rdy** goes high when a stop bit is acknowledged, and goes low on the beginning of the next start bit. **ferr** stays low. data correctly changes when each byte finishes and rdy is high 5) **Requirement 4 & 10** a) Send one byte b) rdy signal c) rdy is high when stop bit is acknowledged and stays high. 6) **Requirement 1** a) Send a stream of data for a defined period of time (10ms) b) No framing error c) **data** at the end is correct to the last byte, **ferr** stays low throughout, **rdy** goes high between each byte 7) **Requirement 1** a) Count time for transmission of a byte b) data transmission at baud rate c) the entire data is received and output within 8/BAUD RATE period 8) **Requirement 1** a) Count how long it takes a bit to be received b) BAUD RATE transfer rate c) all data bits are shown to be asserted for 1/BAUD RATE 9) **Requirement 3** a) Assert data to be low for a small period (less than half a bit period), after a	

	stop bit b) spurious start c) data doesn't change, rdy doesn't change, ferr doesn't change 10) **Requirement 6** a) Send a byte with the last bit being zero, and staying low for two bit periods. b) Framing error c) **ferr** goes high high in the middle of the second zero bit period, and stays high till next start time a low data bit is received	
In submitting this checklist as part of our report, I/We certify that the tests described above were conducted and that the results of these tests are accurately described and represented. I/We understand that any misrepresentation of the tests or the results constitutes a violation of the College policy on academic dishonesty.		
Name(s): Waseh Ahmad & Geoff Watson *Date: 10/02/2017*		

TESTING OUTPUT

TXD-RXD TestBench Output

Testing Simulation-Test-1
Ready is low at the very beginning when no data has been transferred
Received data is high at 0xFF
No Framing Error occurred at the beginning

Testing Simulation-Test-5
New data is ready to be received
Received Data is the same as Transmittd data 10001000
No Framing Error occurred at this transmission

Testing Simulation-Test-4
New data is ready to be received
Received Data is the same as Transmittd data 01010101
No Framing Error occurred at this transmission
New data is ready to be received
Received Data is the same as Transmittd data 00110011
No Framing Error occurred at this transmission
New data is ready to be received
Received Data is the same as Transmittd data 00001111
No Framing Error occurred at this transmission

Self-Checking Receiver bench

Testing Simulation-Test-1
START: checking initial start state ----------
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check

OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
OK: rdy start state check
OK: ferr start state check
OK: data start state check
END: checking initial start state ----------

Testing Simulation-Test-2
START: checking spurious start ----------
OK: rdy spurious start check
OK: ferr spurious start check
OK: data spurious start check
END: checking spurious start ----------

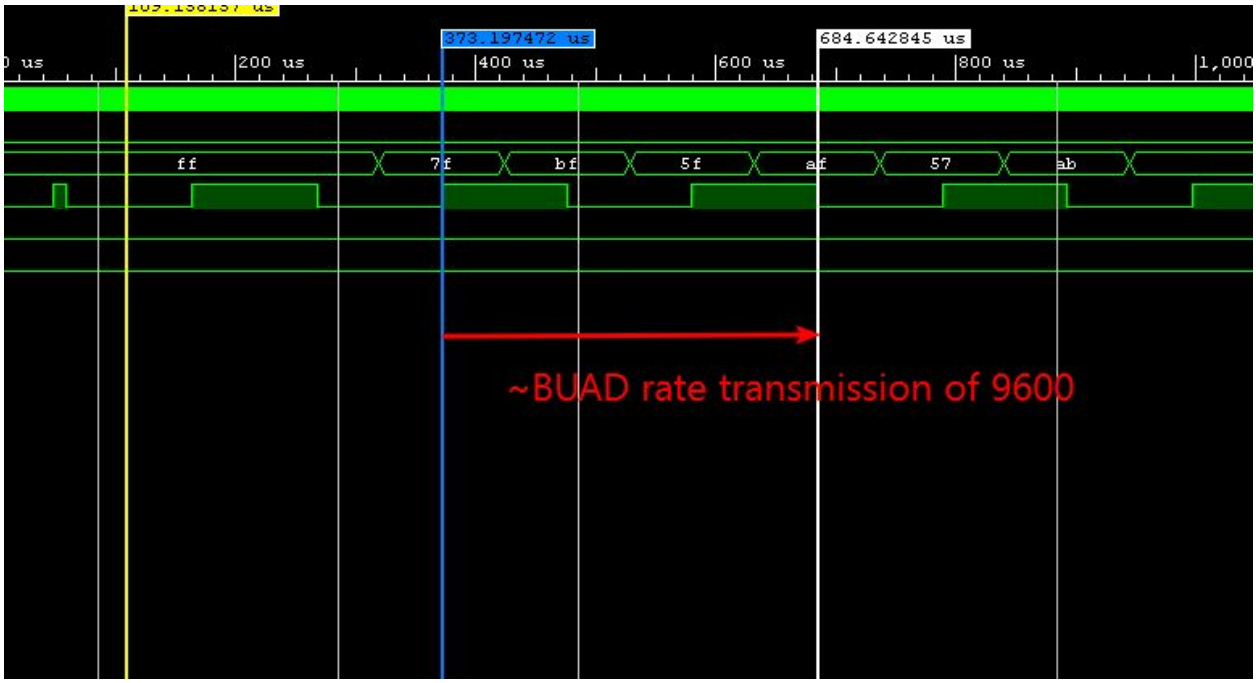
Testing Simulation-Test-5, 7, 8, 9
START: single byte reception ----------
OK: rdy single byte check
OK: ferr single byte check
OK: data single byte check
END: single byte reception ----------

Testing Simulation-Test-6, 7, 8, 9
START: multiple byte reception ----------
OK: rdy multiple middle byte check
OK: ferr multiple middle byte check
OK: data multiple middle byte check
OK: rdy multiple last byte check

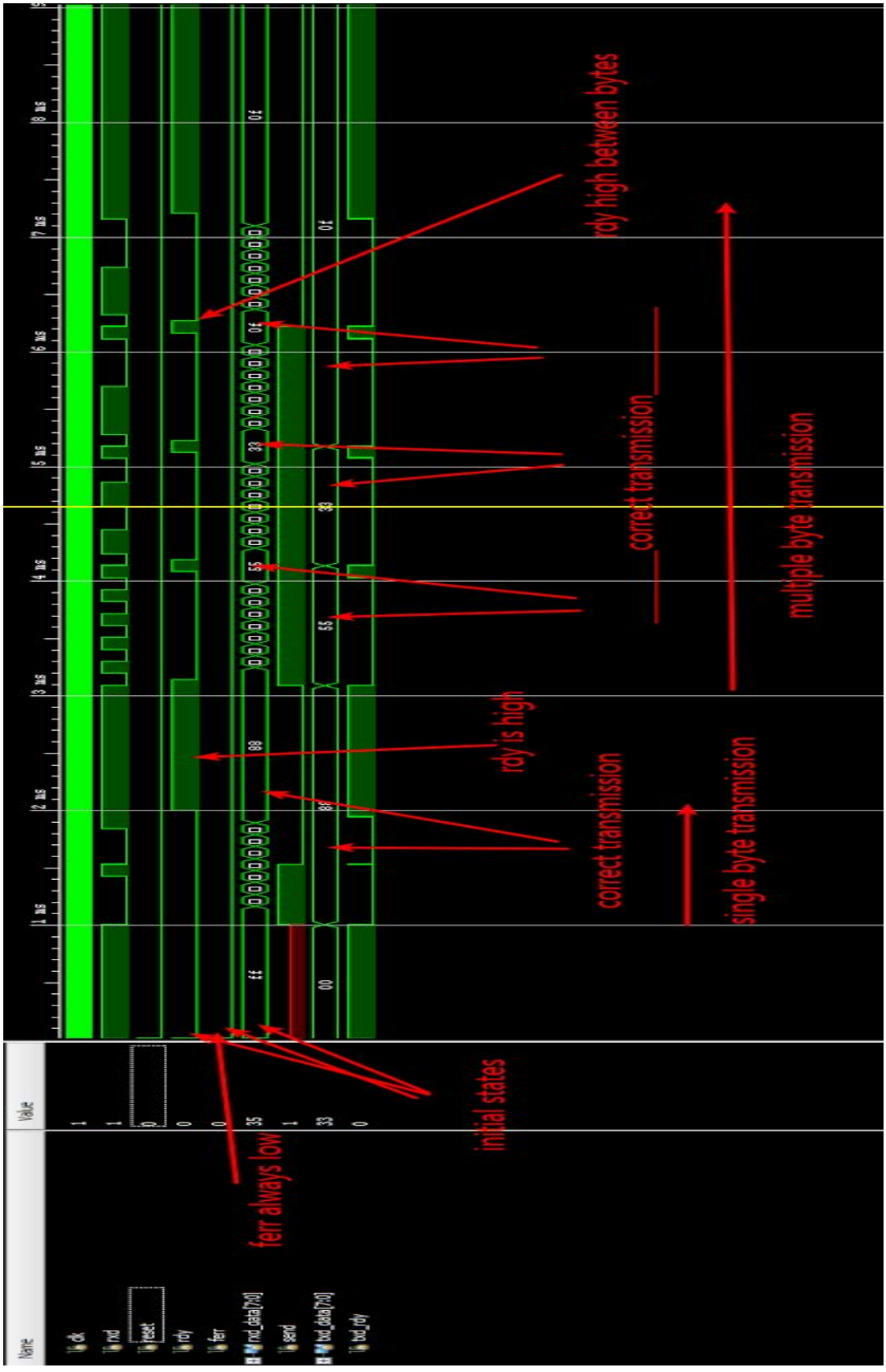
OK: ferr multiple last byte check
OK: data multiple last byte check
END: multiple byte reception ----------

Testing Simulation-Test-3, 10
START:framing error reception ----------
OK: rdy framing error check
OK: ferr framing error check
OK: data framing error check
END: framing error reception ----------

BAUD_RATE SIM_OUT



TXD_RXD_SIMULATION



SELF_CHECK_RXD_SIM_OUT