TECHNICAL MEMORANDUM

Lafayette College

Department of Electrical and Computer Engineering

Title: Manchester Code Transmitter
Author(s): G. Flynn and R. Birru
Date: September 19th, 2016

Abstract

This memo describes how a Manchester Code transmitter based on certain specific requirements was implemented. The design was also to be parameterized to be run at different baud-rates as necessary. A working design was implemented and verified to be fully operational.

1 Introduction

The goal of this memo is to describe one method of implementing a Manchester Code transmitter. Manchester code uses two bits per value when transmitting, with a bit transition in each transmitted value which makes it more convenient than the serial transmitter we designed last week. The design was to send data in 8-bit frames, run off the system clock and be able to be parametrized for use at different baud-rates.

2 The Design

We went through a couple of designs when deciding what to do to implement the transmitter. The first was a state-machine, which recieved data from our existing serial transmitter, drop the start and stop bits and transmit the frame in manchester code.

The second design idea was one suggested during one of Professor Nestor's lectures that employed our existing Serial Transmitter as its nucleus, then use a simple XOR operation with the clock and the serial output to recieve manchester code encoded data. This design seemed exceedingly simpler and this fact guided our final decision to use it.

Due to the specific constraints given to us, we could not employ the Serial Transmitter designed in Lab 2 in its existing state, instead we had to make certain modifications. Several changes were made to the Final State Machine that ran the serial transmitter. The states responsible for sending the start and stop bits were truncated as they were not necessary any longer.

Next, a specific requirement of the Manchester Code transmitter had to be implemented. The transmitter was to send an EOF (End of Frame) at the end of transmission that could be a user-parametrized number of clock cycles. The EOF was employed using a new output '`txen`' that was to remain high for the duration of sending the data frame up until the user-defined wait time was reached. To this end, we added an extra state in the state machine that would hold the 'txen' output high for a certain number of clock cycles after sending was completed. We used the provided parametrized counter module to stay in this state for as long as necessary before either idling or sending the next byte if one existed.

The final modification we made to our serial transmitter was the way our data byte was sent. We previously employed the parametrized counter module to stay in a SEND state until the entire byte was sent. This caused us much trouble during simulation, and we opted instead for a more rudimentary method where we modified the FSM in our transmitter to send each bit in the data byte in its own state as opposed to using a single SEND state. A diagram of the final FSM is available in the the Appendices section of this document.

After the modifications to our Serial Transmitter were made, we used the approach suggested by Professor Nestor and we XORed the modified serial output with the user provided baud rate, which resulted in the serial output being encoded in Manchester Code. A point to note is that we only XORed the serial output when it wasn't idling as the high idle output was not to be encoded in Manchester Code.

3 Design Verification

No.	Description	Test Method	Detailed Results
0	Module Interface ● Check 8 bit data input ● Check 1 bit send input ● Check 1 bit rdy out ● Check 1 bit txd out	Verify module has ports as specified	The module is instantiated with 1 clock and everything else as specified
1.1.1	Check txen high on tx	Self checking testbench will verify	Test passed, see log for more information & Appendix C1
1.1.2	Check txen low @ no transmission	Self checking testbench will verify	Test passed, see log for more information & Appendix C1
1.1.3	Check txen high @ end of bit transmission for EOF_WIDTH bits	Self checking testbench will verify	Test passed, see log for more information & Appendix C2

1.2.1	Check ready high on no tx	Self checking testbench will verify	Test passed, see log for more information & Appendix C3
1.2.2	Check ready low on tx	Self checking testbench will verify	Test passed, see log for more information
1.2.3	Check ready high on 8th bit	Self checking testbench will verify	Test passed, see log for more information & Appendix C4
1.3.1	Verify 1 on txd shape	Self checking testbench will verify	Test passed, see log for more information & Appendix C5
1.3.2	Verify 0 on txd shape	Self checking testbench will verify	Test passed, see log for more information & Appendix C6
1.3.3	Verify idle on txd shape	Self checking testbench will verify	Test passed, see log for more information & Appendix C7
1.4.1	Check data tx on send high	Self checking testbench will verify	Test passed, see log for more information & Appendix C8
1.4.2	Check no data tx on send low	Self checking testbench will verify	Test passed, see log for more information & Appendix C8
1.4.3	Check multiple tx on send high by the 8th bit	Self checking testbench will verify	Test passed, see log for more information & Appendix C9
2.1	Clock wiring • Check all posedges trigger on system clock	Code inspection Using the ‘find all’ feature in Vivado to find all instances of ‘always_ff’ and checking that it was triggered on the 100MHz clock each time.	All triggers are on the posedge of the 100MHz clock
3.1	Check tx: 0xff	Verify each bit via self checking testbench	Test passed, see log for more information & Appendix C10
3.2	Check tx: 0x00 0xFF 0xaa 0x55 0xcc	Verify each bit Verify no gap Verify no glitching	The self checking test bench relies on there being no gap Also the test makes sure that the LSB bit toggles after each test & Appendix C11
4.1	Verify txen on scope	Tx 5 bytes check 4.2ms +/- 100us	See waveform; Appendix B1
4.2	Verify txd multi:	Check bits	See waveform; Appendix B1,

	0xaa 0x05 0x34 0x77 0xff	Check timing Check no gap	B2, B3, B4, B5
4.3	Verify rdy on scope	Tx 5 bytes, check for 3.9ms +/- 100us	See waveform; Appendix B2
5	Baud rate parametrized ● Check baud rate parametrized ● Check system changes when baud rate is changed	All initial tests are run at 50MHz bit rate. The top level is at 10kHz	See Appendix C12
6	Latches ● Verify no latches present	Inspect Synthesis Report for warnings.	No latches found after inspection of Synthesis Report.
	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):Raji Birru & Greg Flynn *Date:09/19/2016*		

4 Conclusion

After demonstrating the hardware it passed all of the testing requirements. The design was exceedingly simple once the modifications to the serial transmitter were completed. Only a single XOR function was necessary to encode the serial data in Manchester Code. We would heartily recommend this approach to design a Manchester Code transmitter after employing our very own design for a serial transmitter.

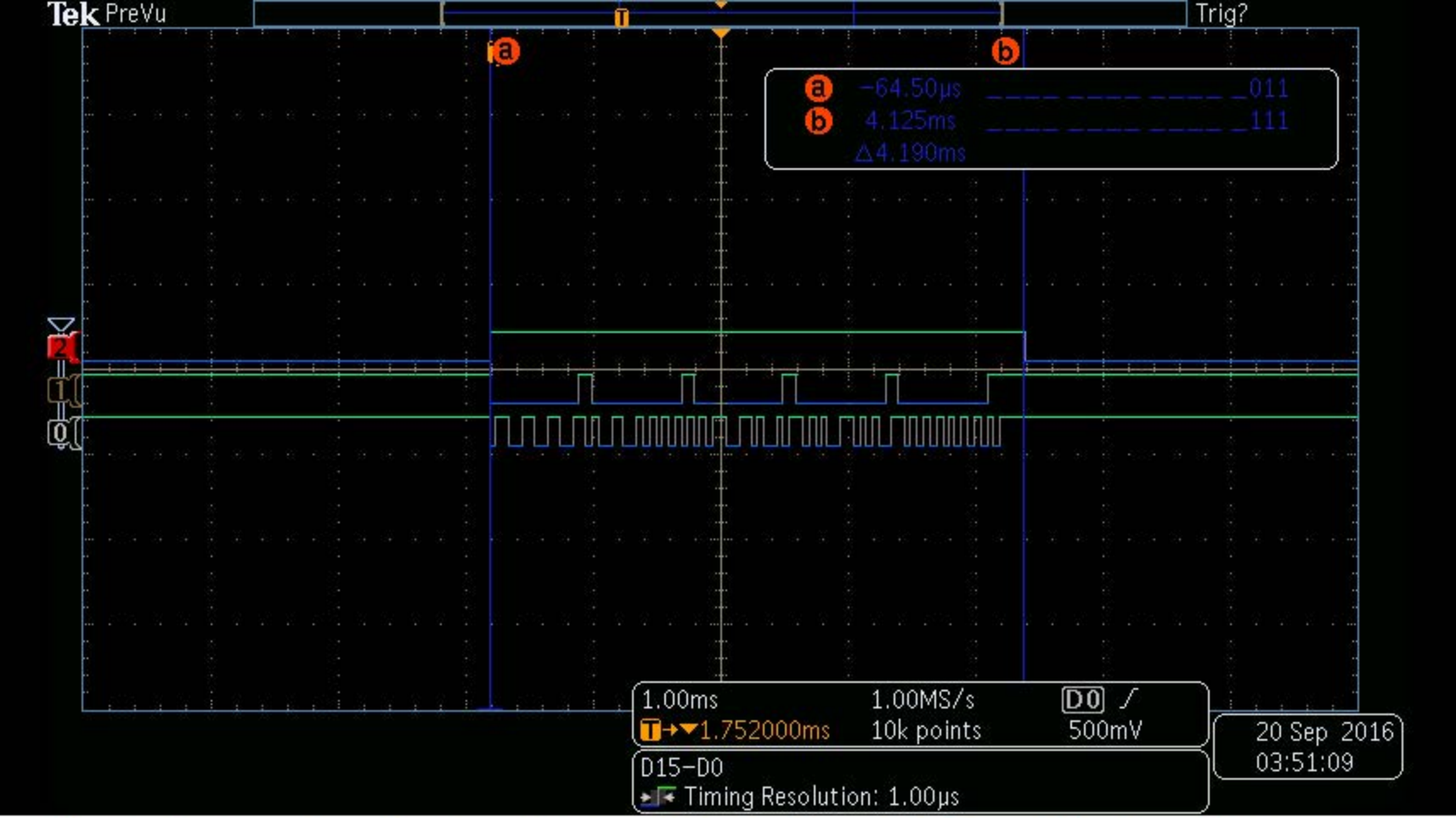
5 Appendices

A FSM Diagram
B Scope Outputs
C Simulation Waveform Outputs

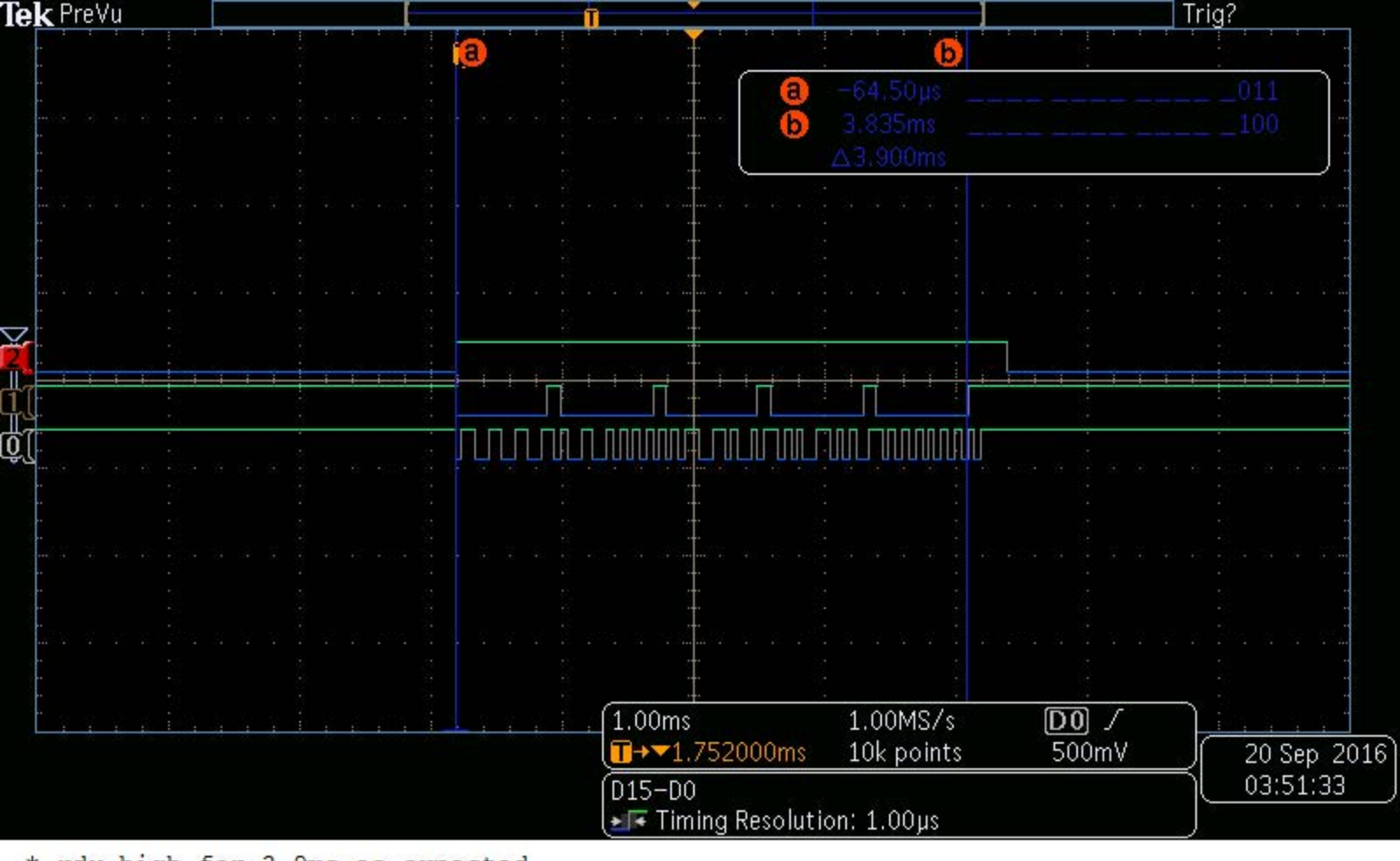
APPENDIX A - FSM DIAGRAM


Reset
IDLE
rdy
txd
counter_reset
eof_reset
idle
clk_reset
send?
no
yes
STAND_BY_START
rdy
txd
counter_reset
eof_reset
idle
lden
enb?
no
yes
SEND_0_LOAD
txd = data[0]
eof_reset
lden
sending
SEND_0
txd = saved_data[0]
eof_reset
sending
enb?
no
yes
SEND_1
txd = saved_data[1]
eof_reset
sending
enb?
no
yes
SEND_2
txd = saved_data[2]
eof_reset
sending
enb?
no
yes
SEND_3
txd = saved_data[3]
eof_reset
sending
enb?
no
yes
SEND_0
txd = saved_data[0]
eof_reset
sending
enb?
no
yes
SEND_4
txd = saved_data[4]
eof_reset
sending
enb?
no
yes
SEND_5
txd = saved_data[5]
eof_reset
sending
enb?
no
yes
SEND_6
txd = saved_data[6]
eof_reset
sending
enb?
no
yes
SEND_7
txd = saved_data[7]
eof_reset
sending
rdy
enb?
no
yes
send?
yes
no
EOF
rdy
sending
txd
counter_reset
idle
eof_count
==
EOF_WIDTH
yes
no

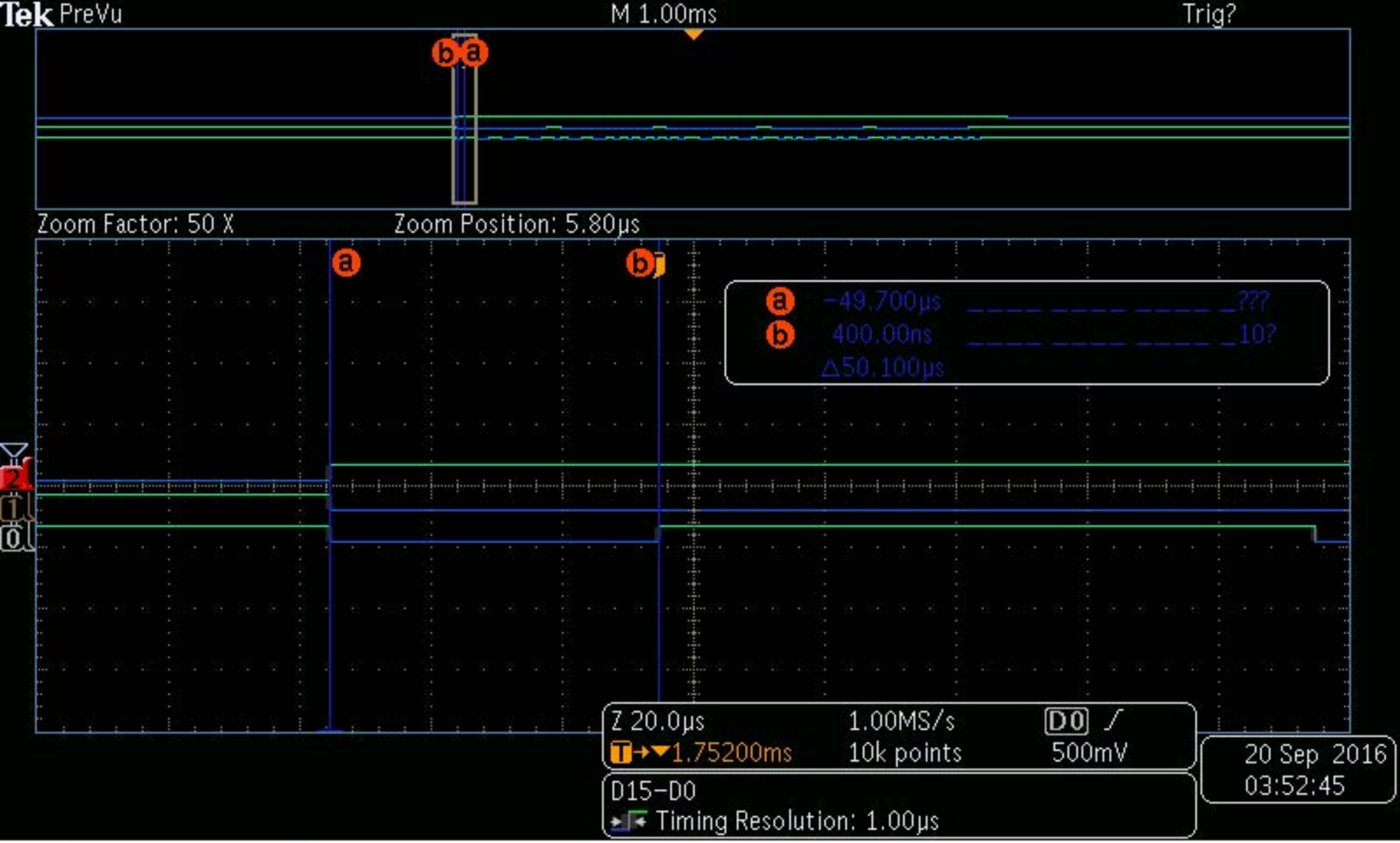
APPENDIX B - SCOPE OUTPUTS



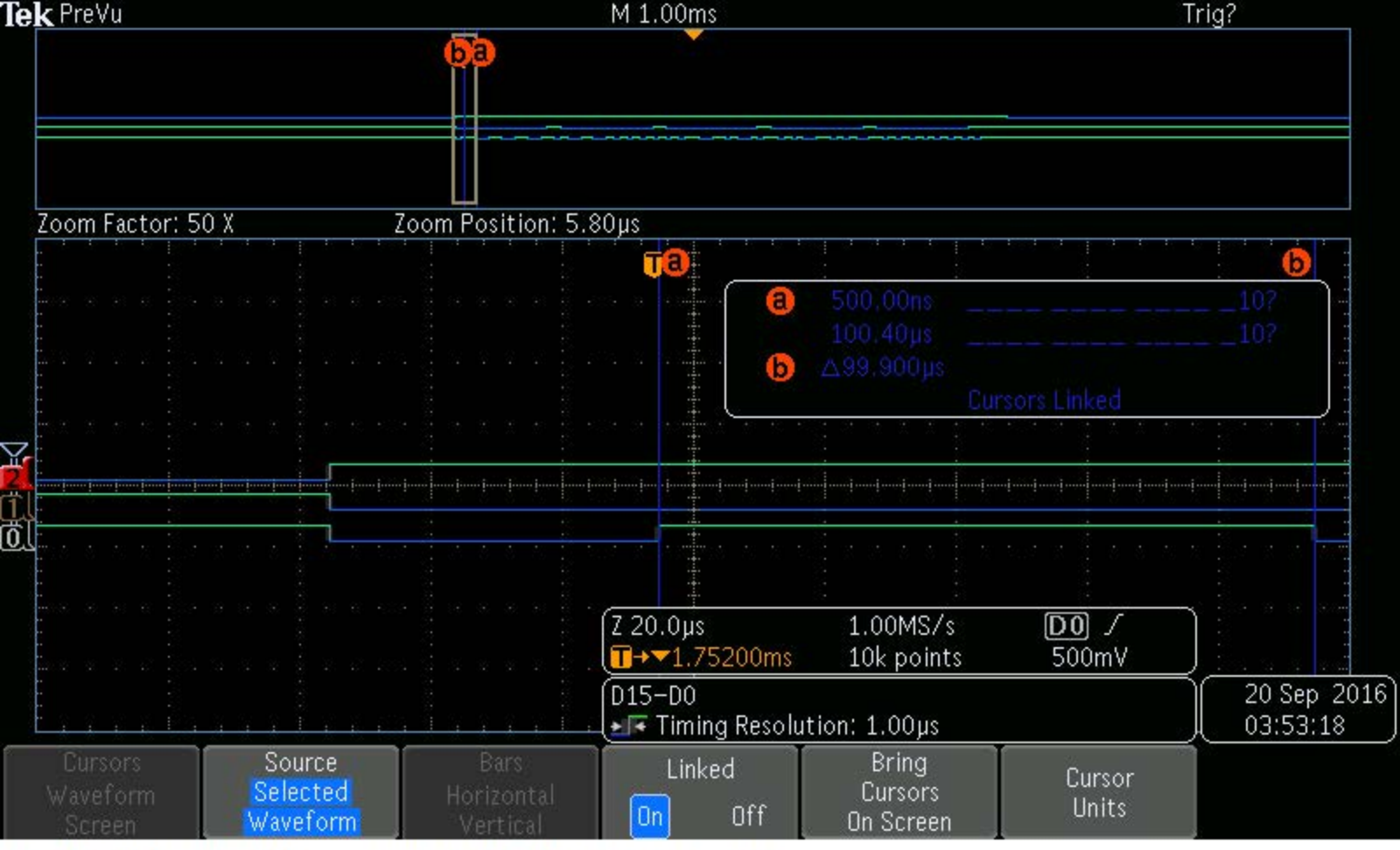
* Sending 5 bits consecutevely. Signals are txen, rdy and txd top to bottom. Notice txen stays high for the duration of transmittion plus EOF duration.4.2ms as expected rdy goes high as bytes are fully transmitted.



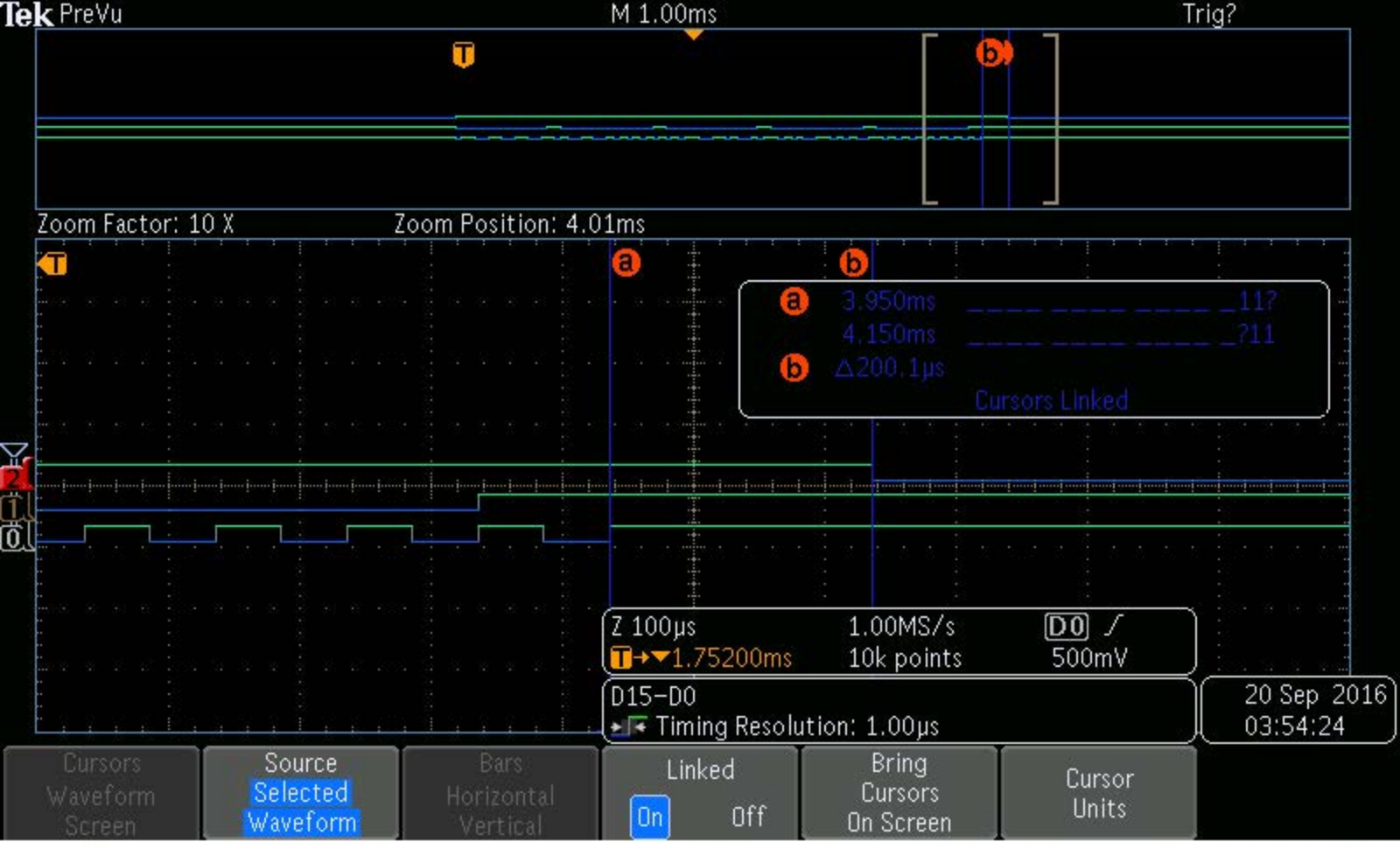
* rdy high for 3.9ms as expected.



* rdy drops when transmission starts. txen goes high when transmission starts. Timing as expected.

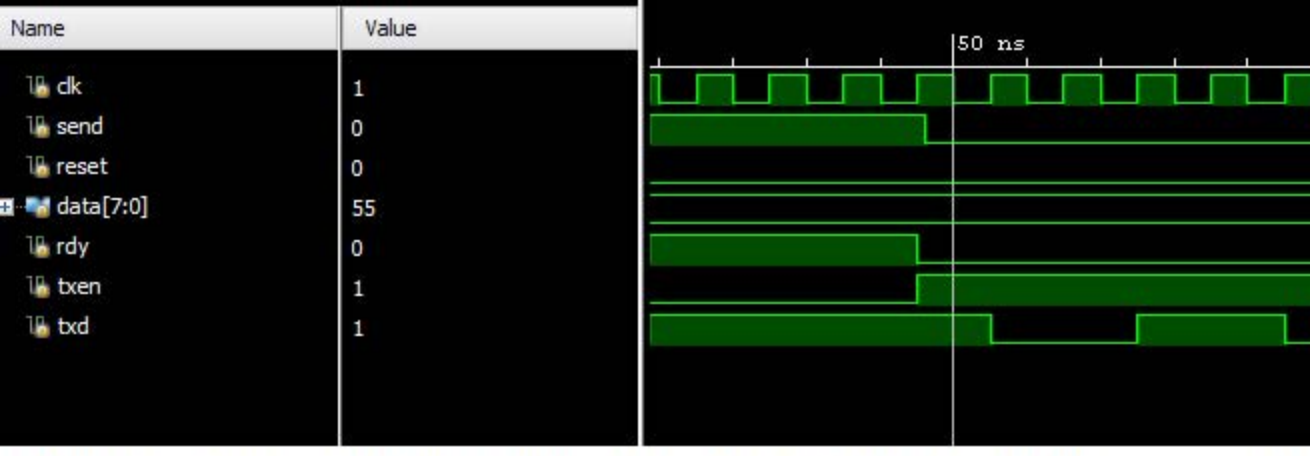


* Bit-width length on scope. Results as expected.

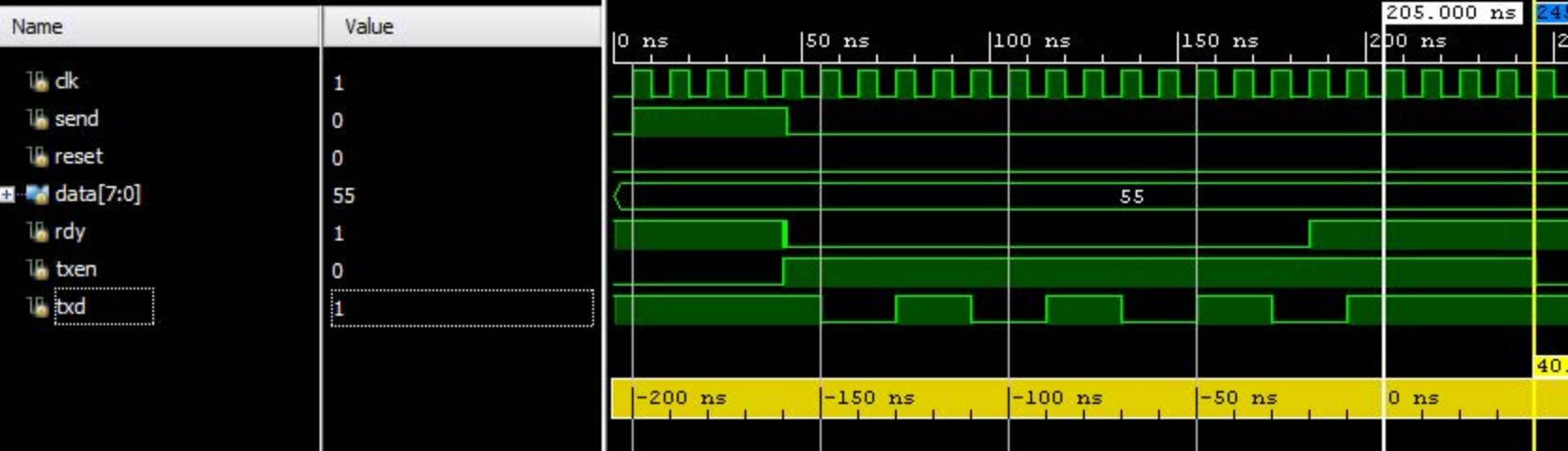


* Timing on EOF on scope. Results as expected.

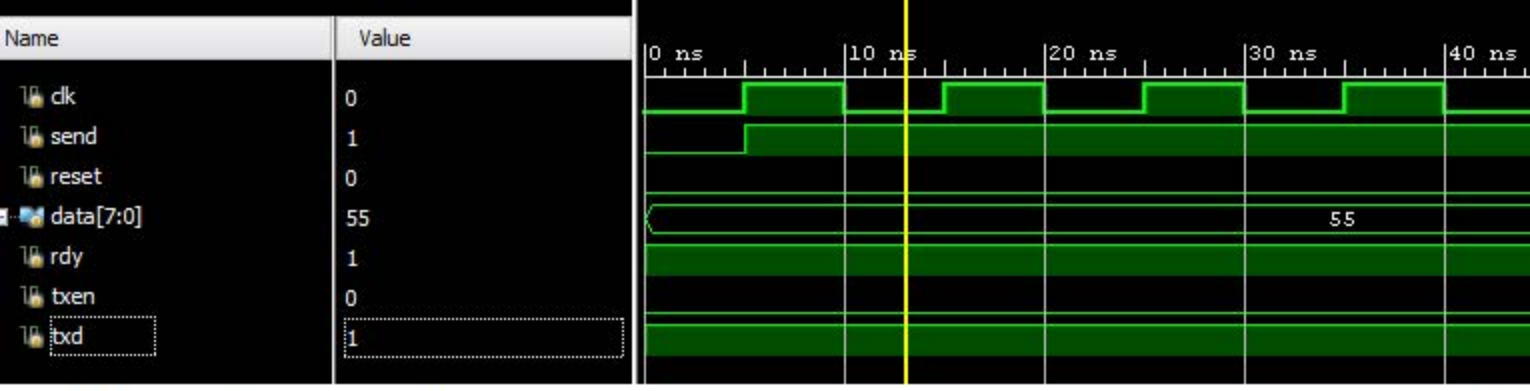
APPENDIX C - SIMULATION WAVEFORMS



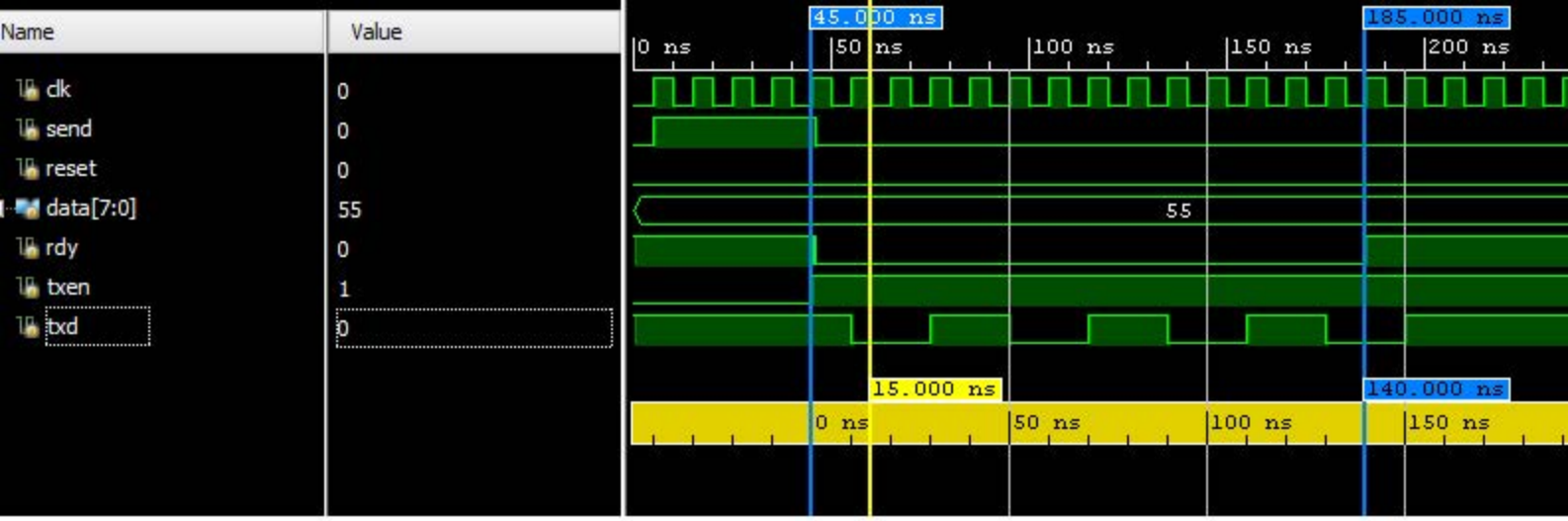
* Notice txen is low before transmition begins and high after.



* txen high after end of transmission for desired amount of time(2 bit-widths). EOF

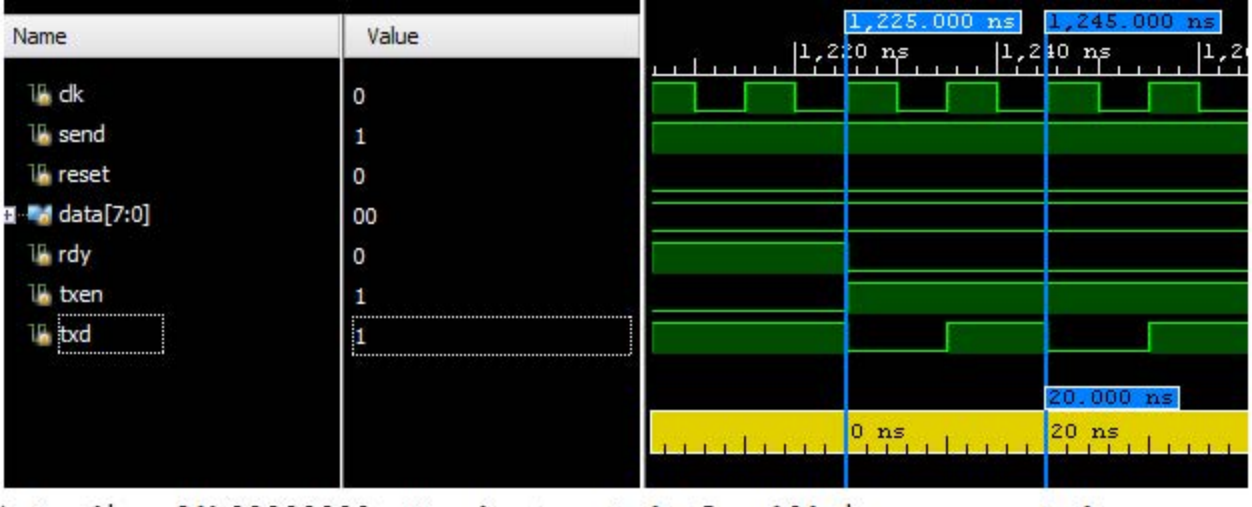


* rdy high when not sending.

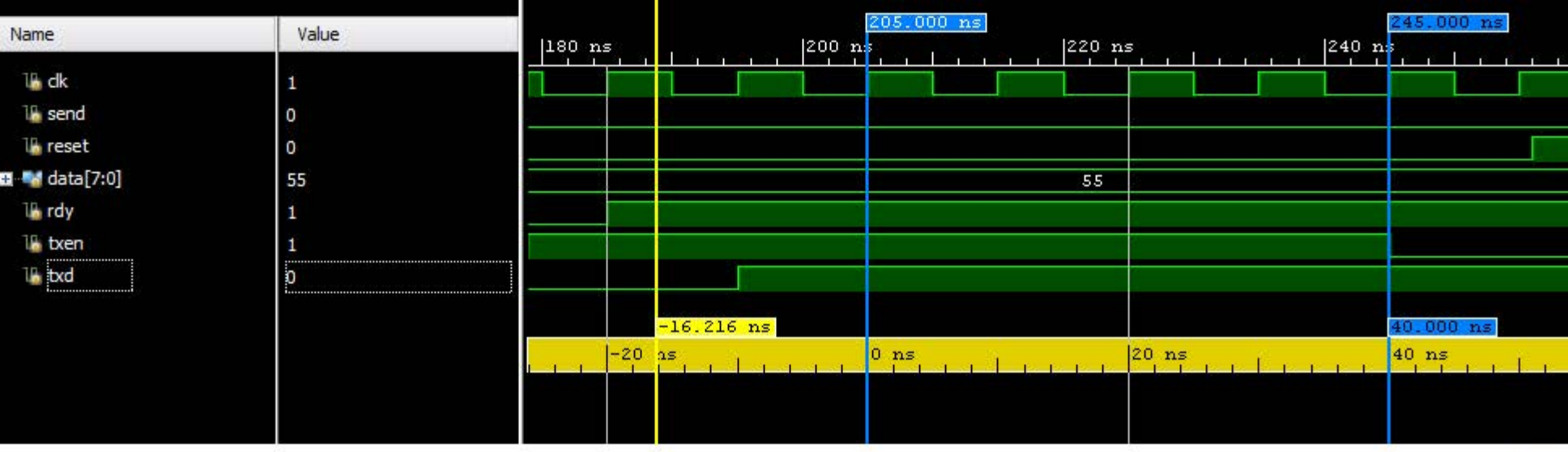


* rdy high when 8th bit starts being transmitted as required.

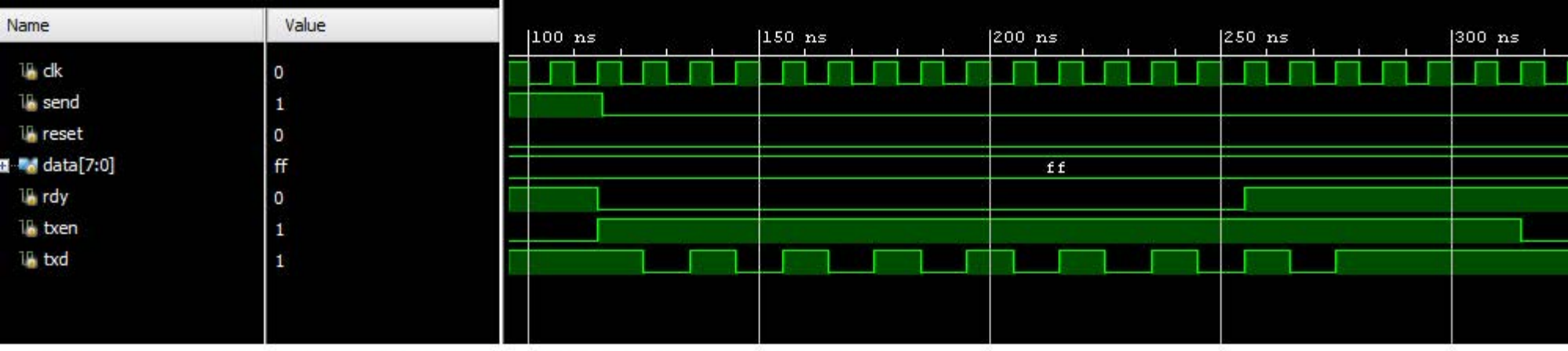
* Sending 8'b11111111. txd output for '1' is as desired.



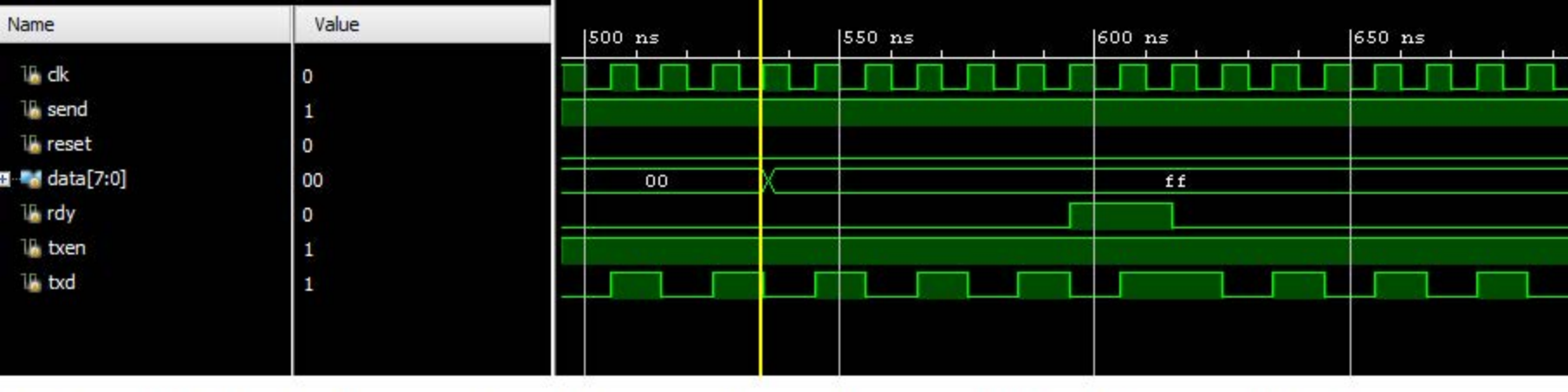
* Sending 8'b00000000. Manchester Code for '0' is as expected.



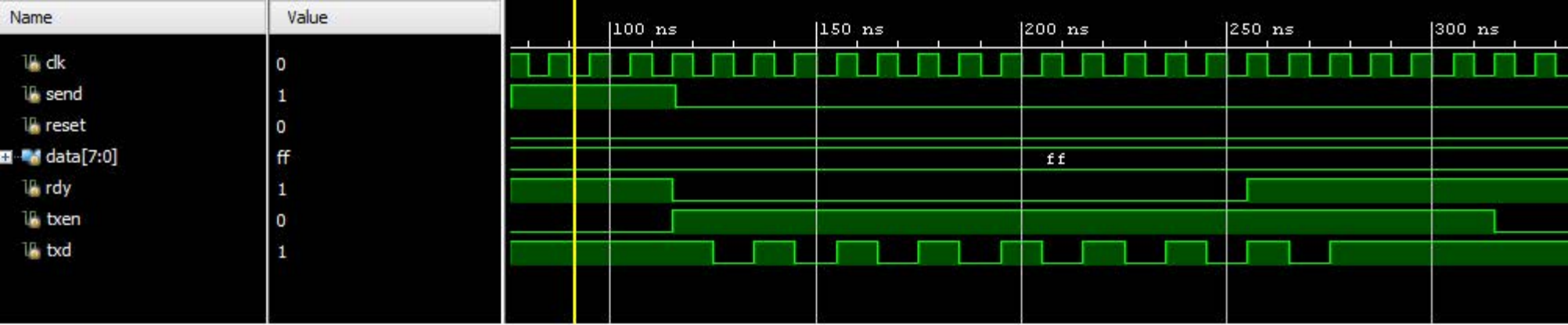
* Nothing being transmitted. txd high as expected.



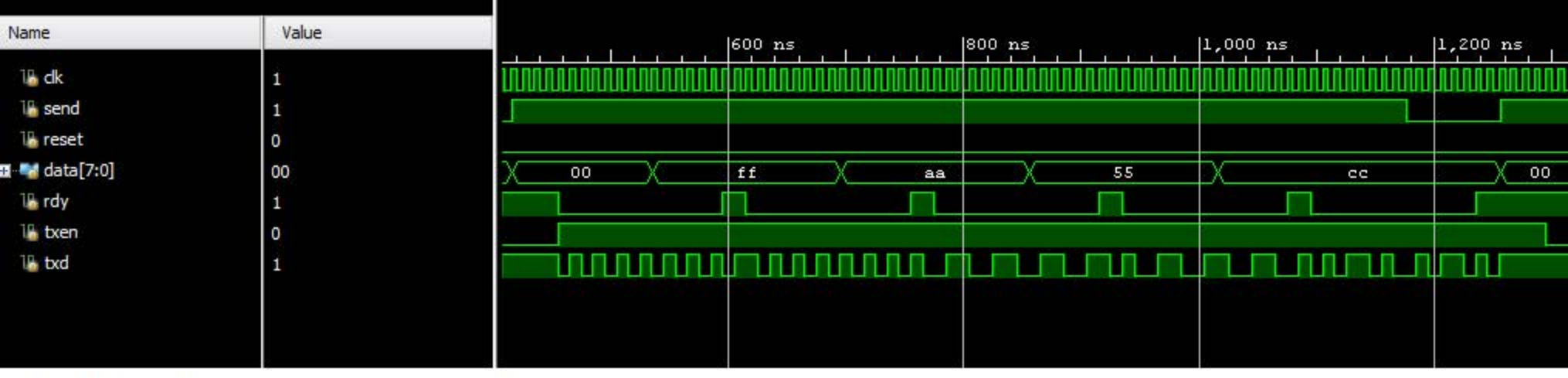
* 0xFF transmitted when Send is high but not when Send is low.



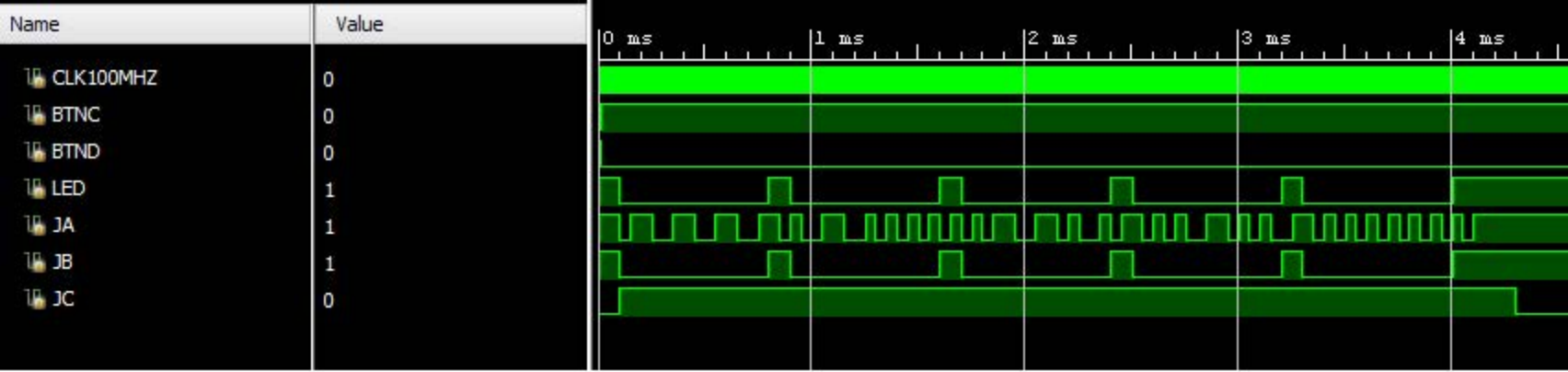
* If send is asserted high as transmission ends, next byte is immediately transmitted as expected



* Sending 0xFF. txd is as expceted.



* Sending 5 bytes in a row. txd results as expected.



* Simulating sending multiple bits from top level.