TECHNICAL MEMORANDUM

Lafayette College

Department of Electrical and Computer Engineering

Title: 7-Segment LED Controller
Author(s): G. Flynn and R. Birru
Date: September 4th, 2016

Abstract

This memo describes how a basic asynchronous RS-232 RTL transmitter was implemented. 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 RS-232 transmitter. For testing purposes eight switches on a Nexys-4 board are used for setting an 8-bit data value to transmit. The data can be transmitted per byte with a button or continuously with a switch.

2 The Design

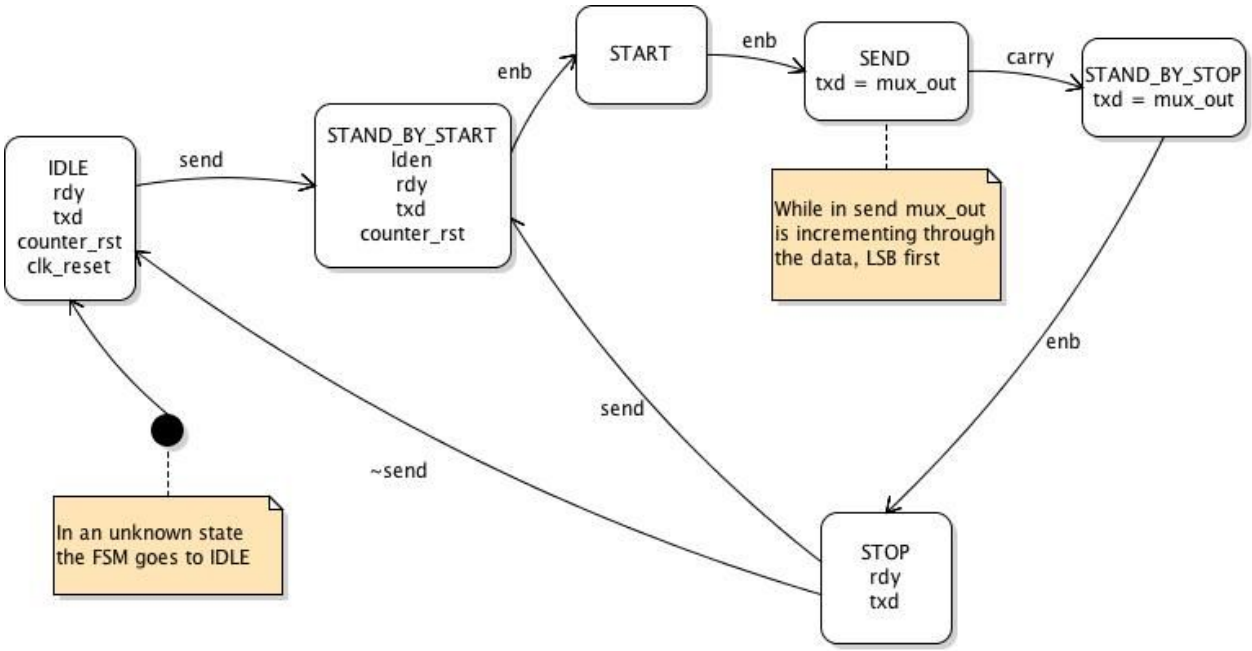
The first design idea we were thinking of employing for our design was a FSM. An alternative idea was a shift register. We ultimately decided to employ a FSM because a it would be much better at handling the rather complex start and stop operations in serial transmission. In addition, for use in future labs, a FSM would be much easier to parametrize and use for sending a number of bits either more or less than the 8-bits required for this specific lab.

We started off by separating the transmission process into steps in our state machine. We had our design begin in an 'IDLE' state where we'd set our output signals `'rdy'` and '`txd`' to high while we waited for a send which would be our signal so start transmission. Due to a need to have our start bit trigger last the specified baud-rate, we employed a standby state before starting sending our data bit. When we get our send, we transitioned into the 'STAND_BY_START' state where the `'rdy'` signal was set low and we waited for our '`enb`' signal that ran off the specified baud to move into the 'START' state where we send the start bit.

For the actual 8-bit data that was to be sent, we used a 'SEND' state where we used an 8-bit multiplexer paired with a counter that would provide the select signal for the mux. The mux was

loaded with the 8-bits to be sent. The counter was setup to count to 9 (It started counting when the start-bit was sent) and selected the respective bit from the mux. It also had a carry output that went high when it was done counting. We used this carry signal as an exit from the 'SEND' state. In case the data is somehow lost after the send signal is sent but before transmission is complete, we saved the input 8-bit data into a register earlier in the 'STAND_BY_STATE' and read it from there.

We then proceeded into the second of our 'STAND_BY_STOP' state which made sure the last data bit was completely sent and after waiting for a pulse from our baud triggered '`enb`' signal to move into the 'STOP' state. The 'STOP' state set 'rdy' and 'txd' high and transitioned into 'IDLE' given that no new data was being sent or started the sending process anew otherwise. A state transition diagram is included right below.



3 Design Verification

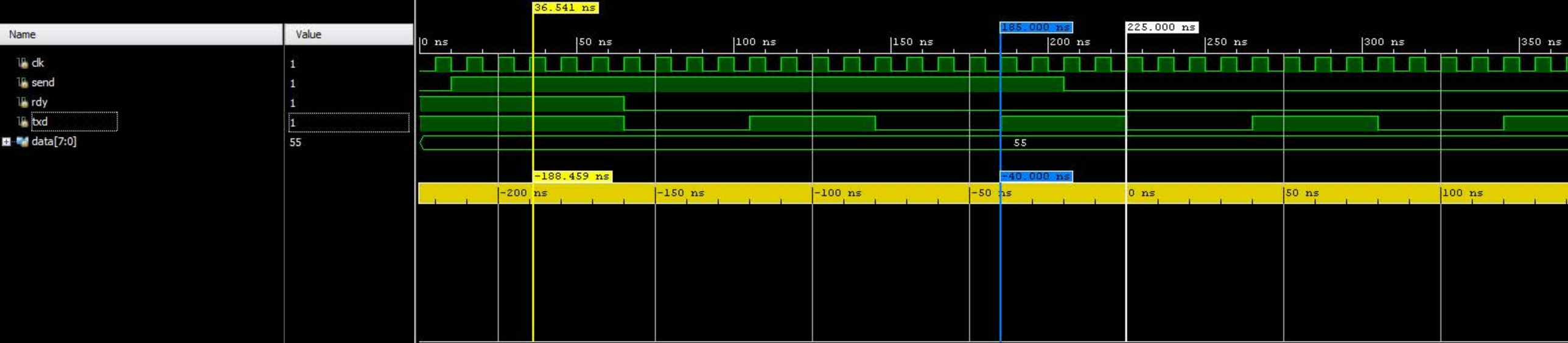
No.	Description	Test Method	Detailed Results
1	Module Interface ● Check 8 bit data input ● Check 1 bit send input ● Check 1 bit rdy out ● Check 1 bit txd out	Verify module has ports as specified	The module is instantiated with 1 clock and everything else as specified

	Send assertion ● Check data transmitted on send assertion	Will be verified in requirement 6	Check results for Requirement 6
2	Ready assertion ● Check rdy drops low as start bit is transmitted. ● Check rdy comes high when stop bit starts	Will be verified in requirement 6	Check results for Requirement 6
3	Multibit transmission ● Check 2 byte frames can be sent ● Check no gap between stop bit and next start bit	Will be verified in requirement 6 as well as requirement 7 and requirement 8	Check results for Requirement 6, 7 & 8
4	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.	After careful inspection, all posedges are verified to be triggering on system clock.
5	Baud rate parametrized ● Check baud rate parametrized ● Check system changes when baud rate is changed	Will be verified in requirement 6 and requirement 7 and requirement 8. Also create a new testbench at a different baud rate	Check results for Requirement 6, 7 & 8 also Appendices A & B
6	Test bench verification ● Send data: ○ 8'b01010101 ○ 8'b00110011 ○ 8'b00001111 ○ 8'b00000000 ○ 8'b11111111 ● Verify multibyte tx ● Also verify no tx on low send	Create 4 tasks, one for each data with the last 2 sent sequentially. Also have a task where nothing is sent. Check all ports are correct Visually inspect data line Check LSB is transmitted first	Check Appendix A-I
7	Communication ● Verify RealTerm receives messages both single/multibyte	Transmit one byte at a time Transmit multiple bits Inspect RealTerm to see messages.	After sending both a singular byte and multiple in succession, all results were as expected when displayed on RealTerm.
8	Scope	Tx 'U' to see one byte. Use 'U'	A 'U' was observed on

	● Verify waveform on scope ● Check timing within 2 significant figures (1/9600 = 104us)	since its ASCII value is 0x55 Use cursors to verify timing	RealTerm. Scope showed correct results and measured baud was 9597KHz for an expected baud of 9600KHz.
9	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:09/04/2016*		

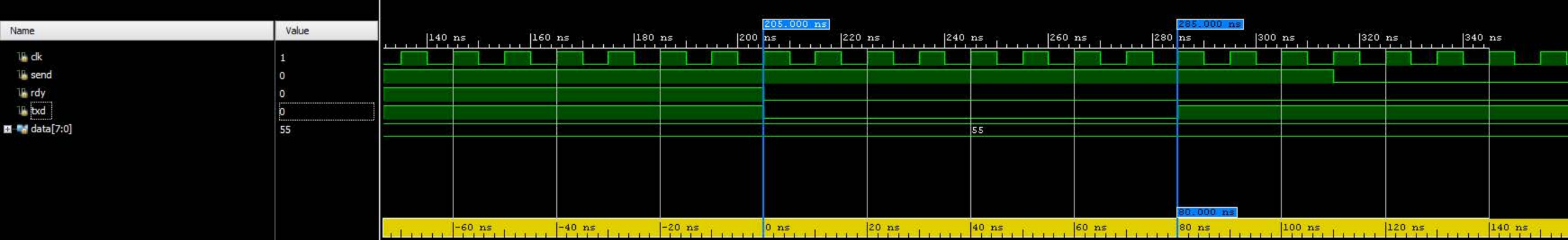
4 Conclusion

After demonstrating the hardware it passed all of the testing requirements. By using the FSM and having clearly labeled state it would be easy to modify the start or stop bit configuration. This flexibility is very useful and we would readily suggest this approach for anyone looking to design a serial transmitter.



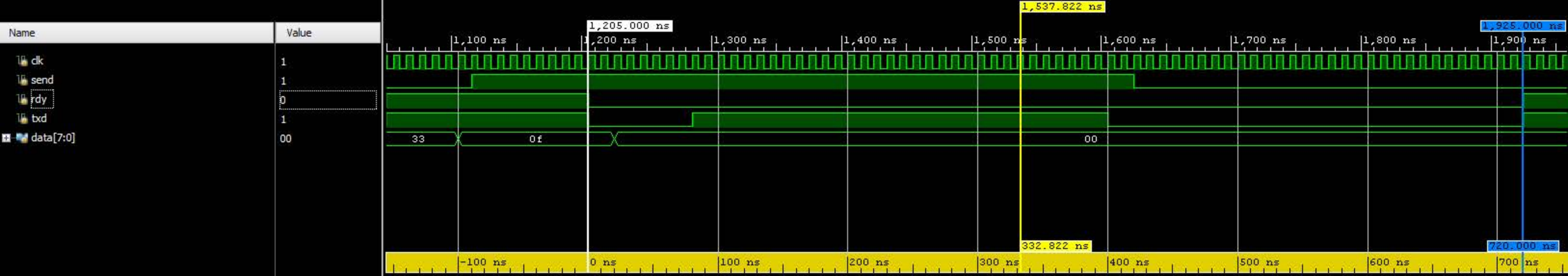
Sending 0x55. We're running tests at a baud of 25MHz and the clock cycle is the expected 40ns long.

Appendix A



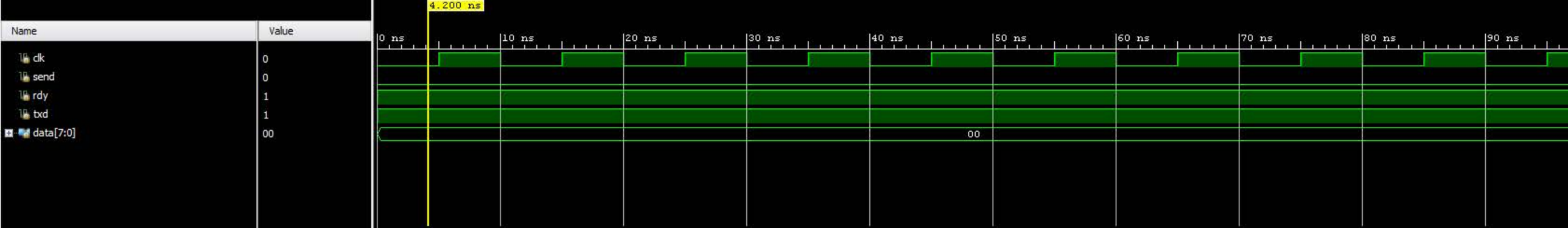
Simulation at a baudrate of 12.5MHz. The duration of a clock cycle is now up to 80ns as expected.

Appendix B

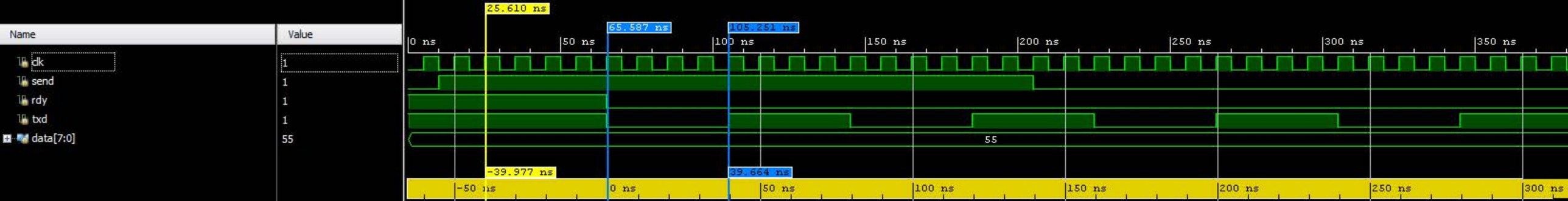


Sending two bits (0f and 00) consecutively. Expected duration is 720ns
and result is the same.

Appendix C

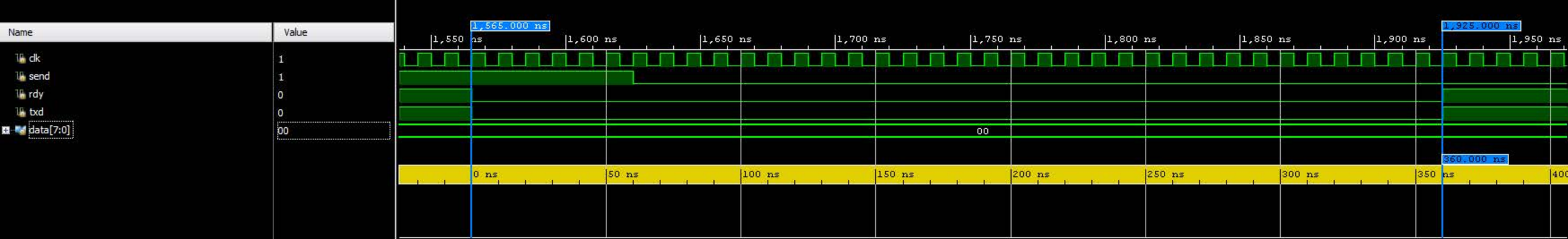

4.200 ns
Name
Value
clk
0
send
0
rdy
1
txd
1
data[7:0]
00
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
00
No transmission when send isn't asserted. txd stays low the whole time.

Appendix D

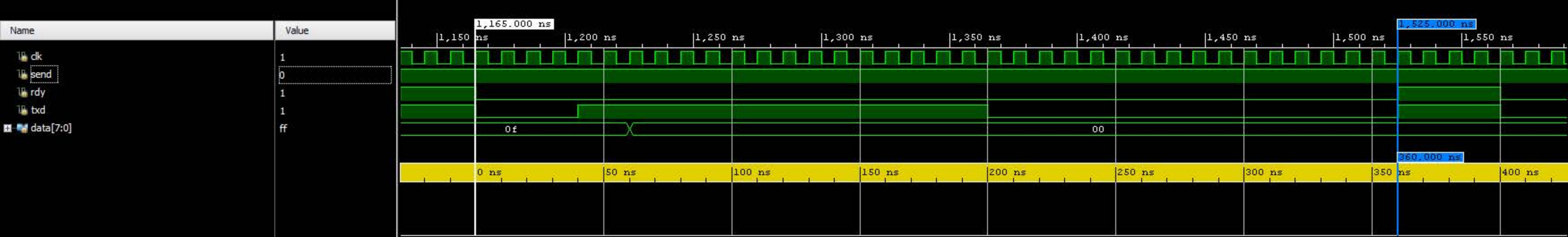


Start bit starts sending after send is asserted.

Appendix E

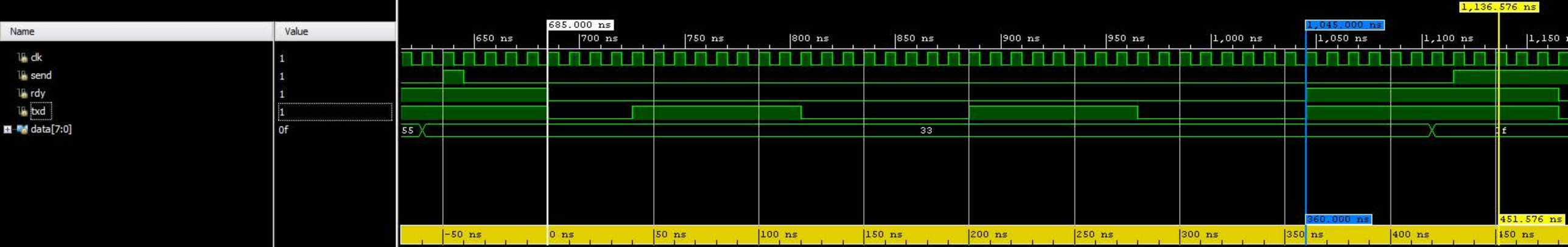

Name
Value
clk
1
send
1
rdy
0
txd
0
data[7:0]
00
1,565.000 ns
1,550 ns
1,600 ns
1,650 ns
1,700 ns
1,750 ns
1,800 ns
1,850 ns
1,900 ns
1,925.000 ns
1,950 ns
00
360.000 ns
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
Sending byte 00. Expected time is 360ns (40ns * 10bits - the stop bit)and result is the same.

Appendix F



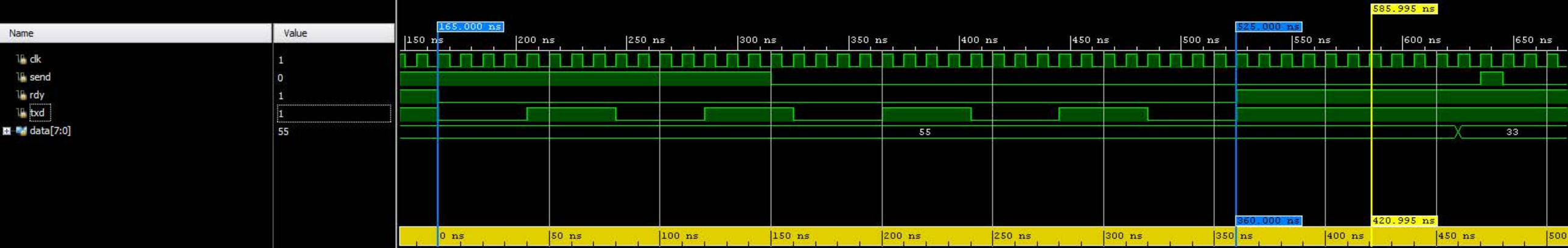
Sending byte 0F. Start bit is easily visible.

Appendix G

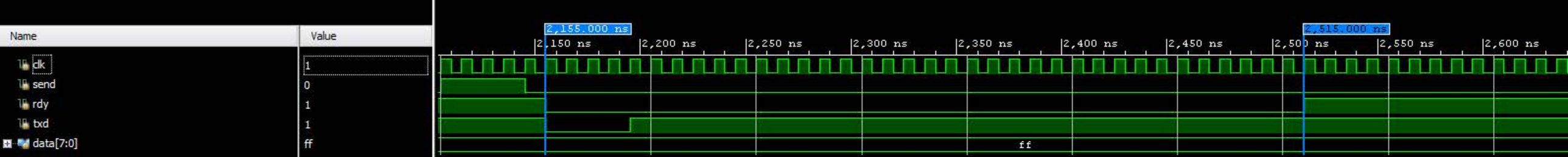


Sending 0x33. Start bit easily visible. rdy goes high after transmission ends. Time duration as expected.

Appendix H


Name
Value
clk
1
send
0
rdy
1
txd
1
data[7:0]
55
165.000 ns
585.995 ns
525.000 ns
360.000 ns
420.995 ns
55
33
Sending byte 0x55. Time duration is as expected. rdy drops low and goes back high when expected. txd results as expected.

Appendix I


Name
Value
clk
1
send
0
rdy
1
txd
1
data[7:0]
ff
2,155.000 ns
2,515.000 ns
2,150 ns
2,200 ns
2,250 ns
2,300 ns
2,350 ns
2,400 ns
2,450 ns
2,500 ns
2,550 ns
2,600 ns
ff
Sending 0xFF. Start bit the only low value as expected. rdy and txd signals responding as expected.

Appendix J