Technical Memo

Geoff Watson & Waseh Ahmad
ECE 491 Lab05 Manchester Receiver
10-31-2017

ABSTRACT

The following report identifies an implementation of the Manchester Encoded 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 a Manchester receiver that can operate at different baud rates and receive frames containing one or multiple bytes. In receiving this data, we must be able to identify when a preamble has been sent by the receiver while simultaneously "locking-on" to the transmission baud rate. In addition, when we receive a complete preamble and sfd, the write signal is asserted after even byte of data.
A detailed description of how the design requirements were met is provided in the test plan and the following simulation output.

Alternative Designs

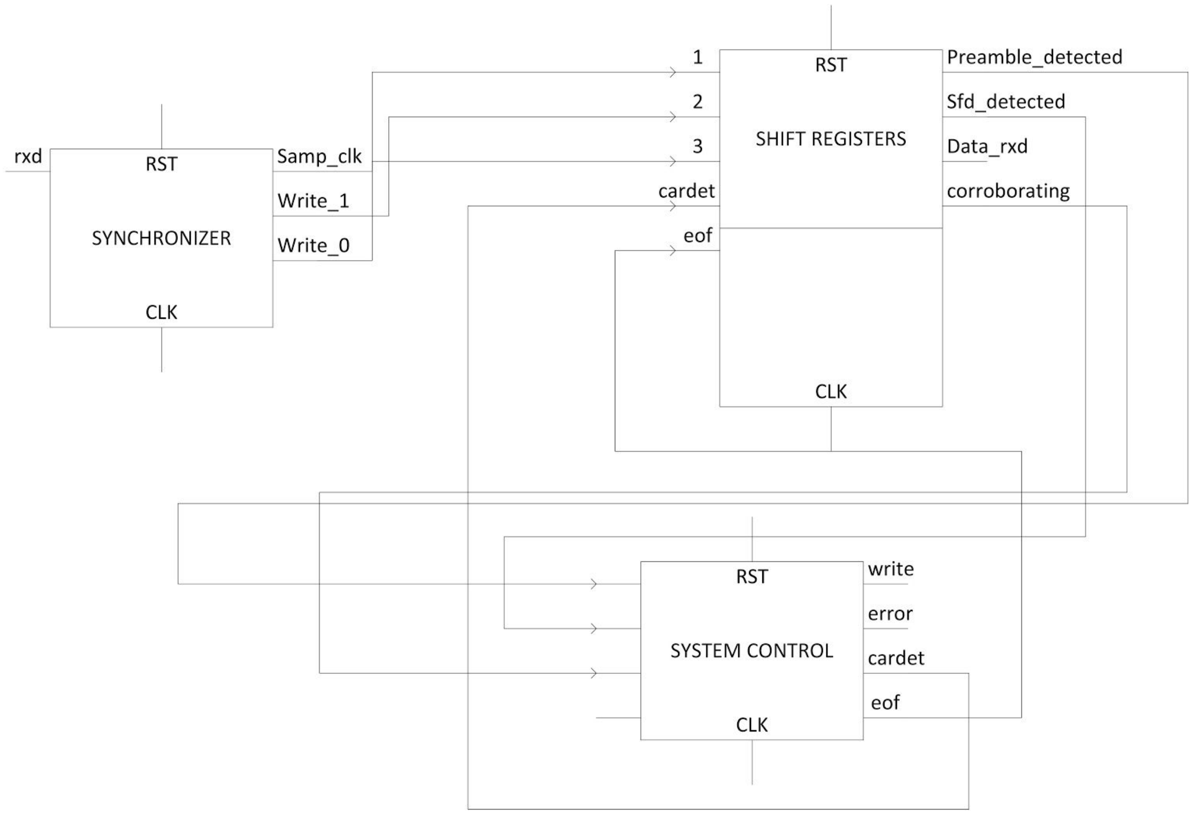
Alternative 1

One of the alternative designs that we considered was one that uses only one correlator to detect whether or not the samples resulted in a bit that was a 1 or a 0. Since the correlator has two thresholds, in our case matching more than 13 or less than 3, an argument could be made that only one correlator is needed to detect what the bit is. We decided against this, however, mainly so that we could use the low threshold (matching less than 3 bits) for pulsing the high threshold (matching more than 13 bits) output. Using this low threshold as a debouncer makes it easier to block out whether or not noise caused a high threshold output.

Alternative 2

An alternate design considered relied on attempting to resynchronize within individual bits i.e. attempt to see how far away the threshold of a correlator is from the actual peak. This design relied heavily on the idea that different frequency input data would result in a bit being recognized as a bit by the correlator at different positions of the rising/falling edge ("01" or "10"). When the threshold for the correlator was reached a function would be used to compute how far away the edge was from the center of the correlator register. Depending on this, the correlator would then emit a speed_up or a slow_down signal to a sampler module. This design was inevitably not considered as it left too much room for noise to control the sampling frequency which negates the benefit of the correlator. In high noise environments, it would become likely that the correlator would interpret a signal as too fast in most cases. e.g. if the following was seen in the shift register of the correlator when the threshold was reached: “01000000 0**1**0*1*1111” the edge would be interpreted as being only 1 bit away (the bold 1) when in reality, it is 3 bits away (italic 1). The correlator would determine the data was coming in too quickly and tell the sampler to slow down. To fix this, it would be required to check for the edge in a bigger range of consecutive 0s and 1s *within* the correlator register. This would make the system more complex, less robust and with greater chance for errors.

Chosen Design



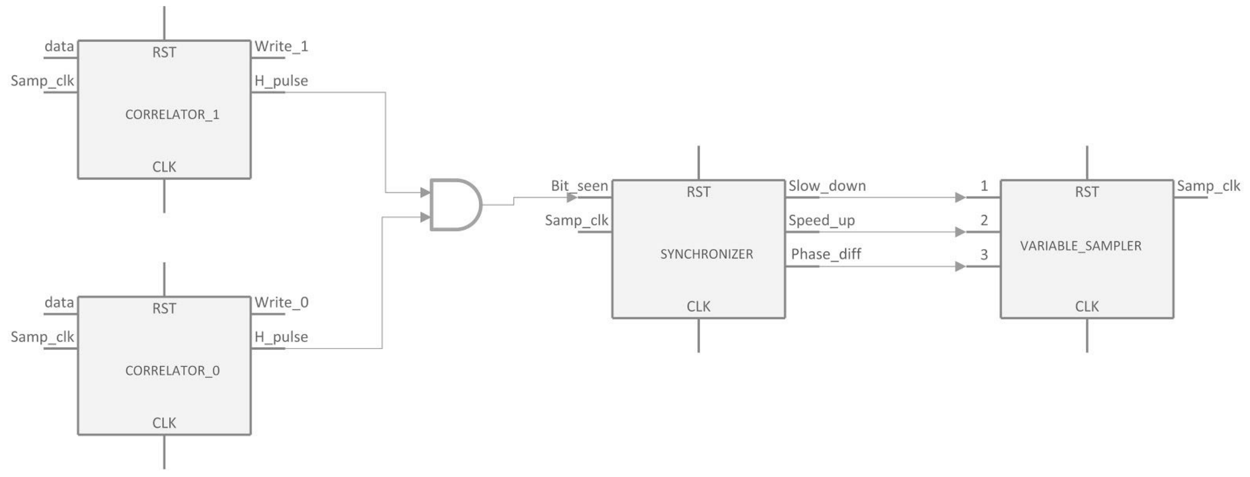
High Level Block Diagram

This design was chosen as it allowed for the greatest modularity of blocks without making one block too complicated. It also allowed synchronization without too much interference from white noise. It still has downsides such as interpreting heavily corrupted data which might throw off the synchronization. This might not necessarily be bad as in case of corruption, whether or not to even read data may or may not be ideal.

Our chosen design is presented below. Its main features include:

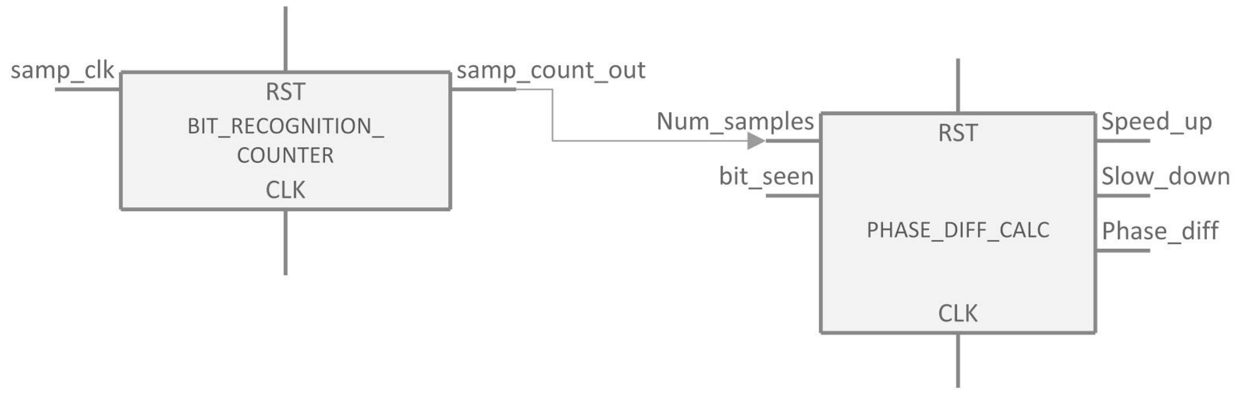
1. Synchronizing stage which is mainly responsible for maintaining synchronization of the received data.
2. Registers to store data as interpreted from the synchronizing system.
3. FSM for the pre-data stages
    a. Locking onto preamble
    b. Recognizing SFD
4. FSM while receiving data
    a. Recognizing and writing bytes of data
    b. Recognizing when an error takes place

The design takes the following procedure:
As data is received by the receiver, it is input into the correlators. The correlators use a sampling clock to sample the data. The correlators are used to provide a margin of error with the presence of noise. Once a correlator estimates the presence of a rising/falling edge, it asserts that it has seen a bit and then that bit is passed into the preamble register. At the same time, the synchronizer keeps track of how many samples had to be taken to actually have that bit recorded in the register. If the number of samples taken to see the bit was too long or too short(by default, relative to 16), the synchronizer will signal the variable sampler to change its sampling frequency in order to attempt to synchronize with the falling and rising edges. Once the preamble is detected i.e. 8 bits of alternating 1s and 0s, cardet is asserted and the system can then start also looking for the SFD. If the SFD is recognized, incoming rxd can then be interpreted as data. If an error occurs during SFD recognition, cardet is asserted low again and the system goes back to trying to lock onto the preamble (note that while trying to find SFD, the preamble will also be constantly verified). When the SFD is verified and data is now being received, the FSM keeps track of how many bits have been seen, asserting write after each byte. If an error occurs before a full byte i.e. no rising or falling edge (EOF bit or inverse), the error signal is asserted and the system resets to trying to find the preamble. If however, the EOF is seen after an integer multiple of bytes are seen, the system recognizes the end of the frame and resumes to trying to find a preamble.



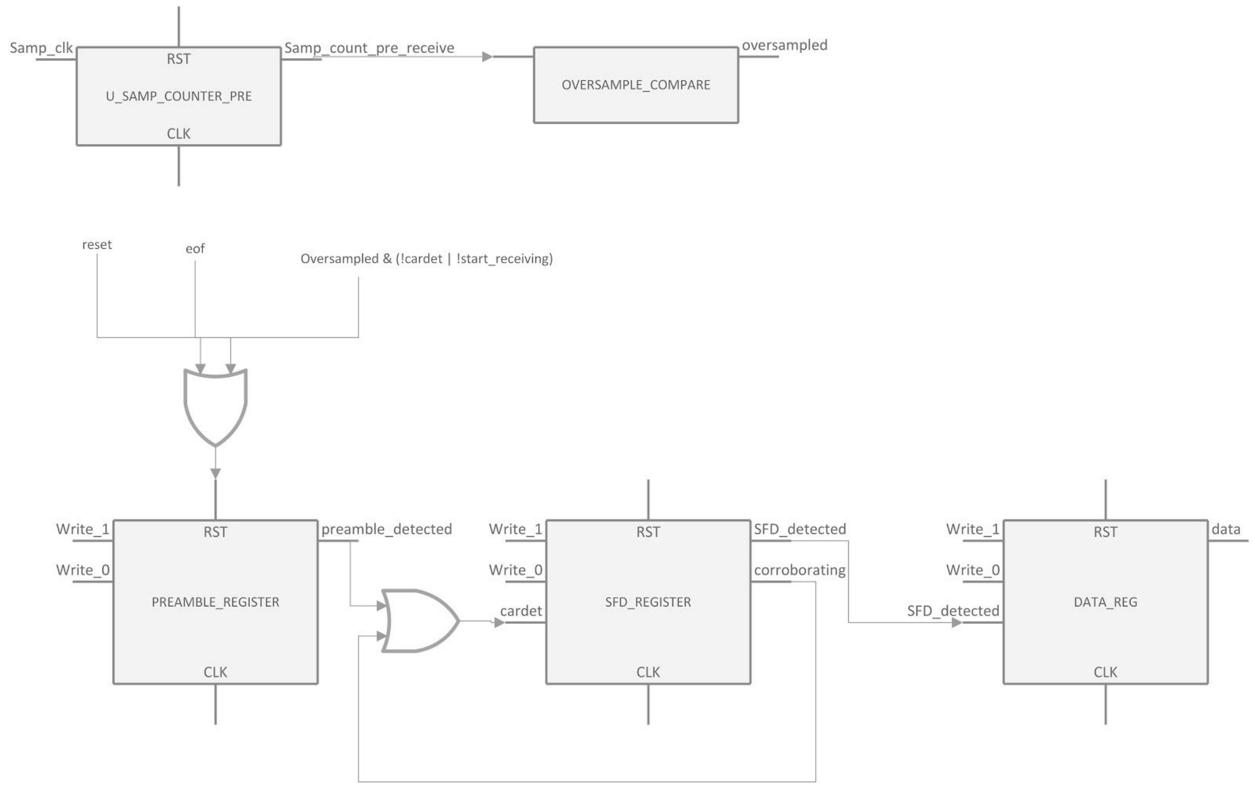
Synchronizing System Block

The synchronizing system block shows the two correlators for recognizing falling and rising edges and how they are connected with the synchronizer. When bits are assumed to be seen by the correlator, the synchronizer calculates whether the bit took too many samples or too few samples to be seen and signals the sampler to speed up or slow down. It also provides the sampler with the magnitude of the assumed phase difference to result in a proportional feedback.



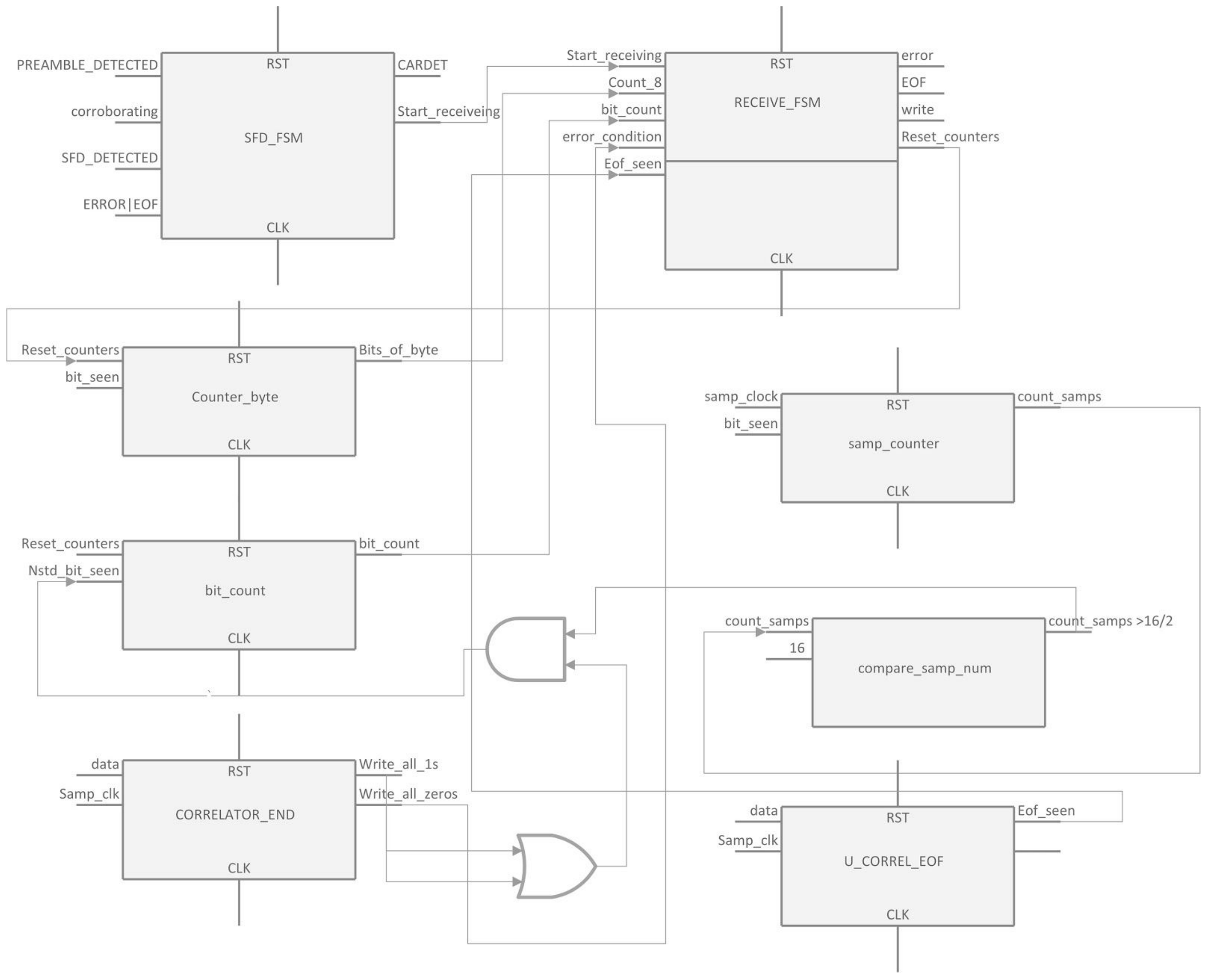
Synchronizer Block

The synchronizer block contains a counter and a phase difference calculator. The counter is enabled by the sampler signals. The output of the counter is compared with the value of the length of the length of the correlator pattern (default 16) only when a bit is seen. This comparison outputs whether the sampling needs to slow down or speed up in order to synchronize.



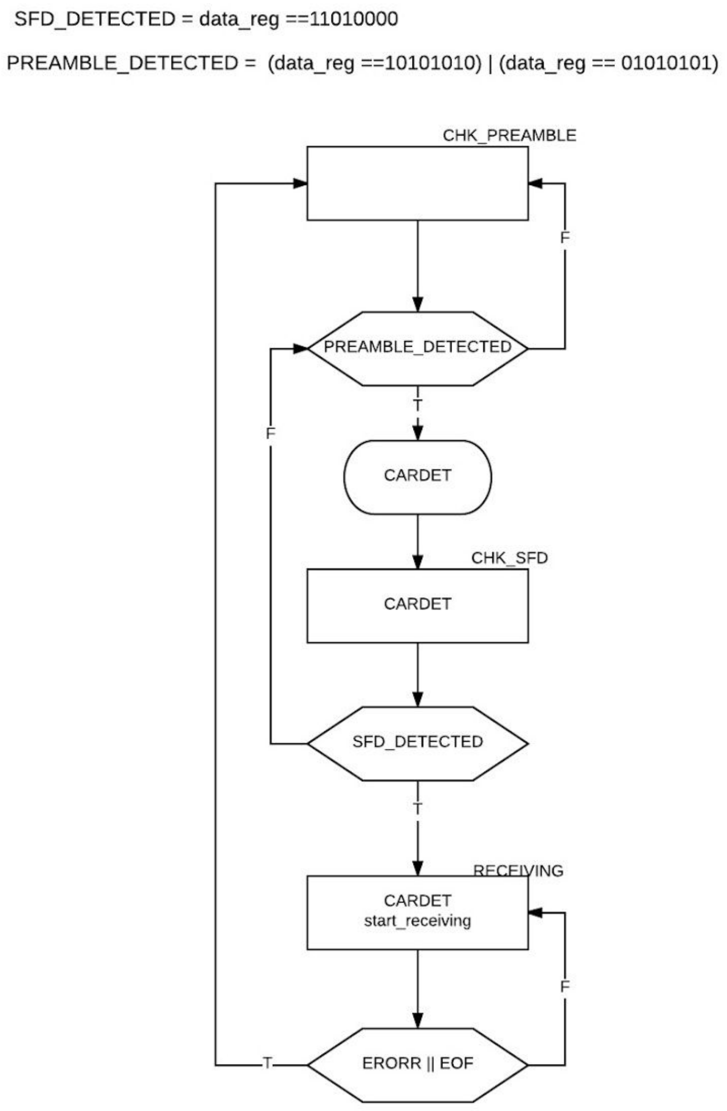
Registers Block Diagram

The shift-register blocks are used to store appropriate data as each bit is recognized. Initially, the PREAMBLE_REGISTER is the only register storing 8 bits of data at a time. Once 8 alternating 1s and 0s are seen, the cardet can be asserted high and the SFD_REGISTER can start accepting bits as well. The SFD_REGISTER (shift-register) also maintains signals that tell whether the incoming data is corroborates with the SFD pattern or if the SFD is fully detected. Once the SFD is detected, the DATA shift register can start accepting data as well.



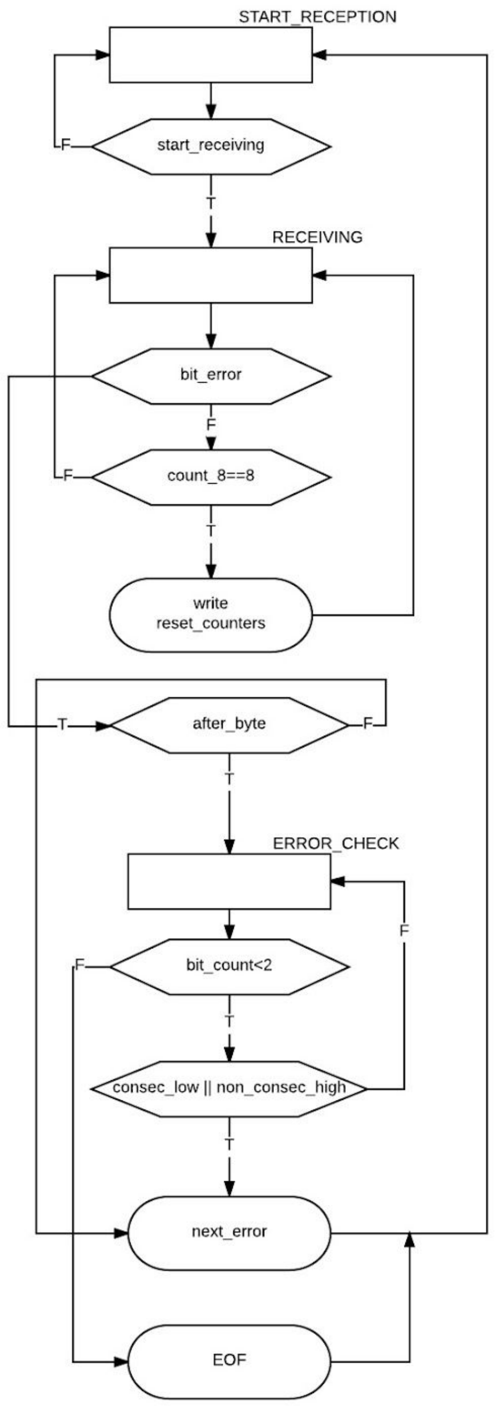
Finite State machine blocks with error/EOF check

In the diagram above, the FSMs are described with their associated counters. One counter is used to count the number of bits seen to form full bytes. Another counter is used to count the number of consecutive HIGH or LOW signals seen. These signals are provided by a correlator which has a pattern of 16’hFFFF. The correlator provides outputs to test for occurrence of errors or EOF signals.



State diagram for bit reception stages

The state diagram above shows the FSM used for recognizing SFD and the PREAMBLE. Until the preamble is detected, the CHK_PREAMBLE state is maintained. After which, the cardet is asserted and the system goes to the CHK_SFD state. There, the FSM continuously checks for either the PREAMBLE or the SFD. If neither is found, the system returns to the CHK_PREAMBLE state. When the SFD is verified, the system goes into the RECEIVING state which signals the other FSM to proceed.



ASM diagram for receiving data bits

The diagram above depicts the state diagram after the SFD is verified. The system continuously checks for an error signal or a full byte. If a full byte is seen, the system outputs a write signal. If an error occurs i.e. consecutive highs or consecutive lows, the system proceeds to the ERROR_CHK state. There, the system determines if an error is occurring or if an EOF is being seen. After that is determined, the system returns to the initial state.

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, realterm and simulation. Connect with manchester transmitter for hardware demonstration.	Data transmitted from the Nexys 4 DDR board shows up of the 7 segments and in RealTerm 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. Cardet 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 the **man_receiver, receive_fsm** and **sfd_fsm** files
4. Contains no latches	Inspection of Synthesis Report	Synthesis report contains no latches reference.

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. Receiver shall be implemented in a module named mx_rcvr with inputs **rxd, reset, clk**, and outputs **cardet, data, write,** and **error** 2. Timing diagrams must match the ones from the description. 3. Receiver should ignore all random input changes until an incoming preamble indicates the start of a frame. Preamble will have a minimum of 16 alternating manchester 1's and 0's (but must be able identify preamble with 8 minimum, maximum of infinity). After the preamble the SFD will follow with a constant pattern "11010000", LSB first. 4. Once the SFD is detected, accept all data bits and output them as bytes until the incoming data returns to the "idle" state. 5. Receiver will recognize the EOF and ignore random input following 6. receiver will recognize an error when (1) the input is low/high for consecutive halves of a transmission or (2) when a transmission ends before receiving a full byte. Error is asserted high when	Format → Test--object being tested-- how to check for Pass **Hardware** **1) Requirement 1, 2, 9** a) Press the reset button b) System reset c) **Cardet** and **error** LED are low, **data** = all zeroes **2) Requirement 12** a) Send a frame with one byte of data using mxtest b) Correct Reception of one byte c) byte displays on 7-seg display in hex(least sig. positions) and in RealTerm, **carder** goes low after byte, **error** is LOW **3) Requirement 12** a) Send a frame with multiple bytes using mxtest b) Correct reception of multiple bytes c) all bytes are displayed in RealTerm, with the 4 most recent ones being shifted left on the 7-seg display after each new byte, **cardet** is high during reception then goes low, **error** is low throughout 4) **Requirement 7, 9** a) Increase the transmission rate on the transmitter and try sending a frame b) Correct reception of multiple bytes c) all bytes are displayed in RealTerm, with the 4 most recent ones being shifted left on the 7-seg display after each new byte, **cardet** is high during reception then	All tests PASS using hardware and simulation as shown below. BAUD RATE 50000(default) Hardware tests verified PASS by Professor Nadovich

either of these cases is true until a new frame BEGINS or the circuit is reset. 7. While receiving data, the receiver design shall continuously re-synchronize using the transition of the manchester bit to adjust timing as required. Need to be able to receive frames in which the actual transmitted bit rate varies from the intended bit rate by 1% 8. Cardet shall be asserted as soon as it "locks on" to the preamble (at max 8 bits into preamble). 9. Receiver needs to be parameterized for different a BIT_RATE. (instructions on how to change need to be in the header). 10. Need one or more self-checking testbenches that cover: (a) 24-bit preamble/SFD followed by a single byte of data (RXD high before and after frame) (b) 24-bit preamble/SFD followed by 24 bytes (RXD high before and after frame). (c) 24-bit preamble/SFD followed by a single byte of data (RXD is randomly varying for 10^6 bit periods before and after frame). Need to only receive one frame and have no spurious starts. To generate a random number between 0 and n, use {$random } % n (d) deliberately wrong input where RXD is held low for an entire bit	goes low, **error** is low throughout **Simulation** 1) **Requirement 1, 2, 9** a) Assert reset b) initial states c) **cardet**is low, **data** = 8’hxx, **error and write** is low 1.5 a) Send one and a half preamble b) Cardet c) Cardet goes low after last half of preamble 2) **Requirement 4, 7, 8, 10 (a)** a) Set **RXD** high then send 24-bit preamble/SFD followed by a single byte of data and EOF then set **RXD** high b) Reception of one frame **c)** **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **data** gets set to the new value after EOF, **error** is low 3) **Requirement 4, 7, 8, 10 (b)** a) Set **RXD** high then send 24-bit preamble/SFD followed by a 24 bytes of data and EOF then set **RXD** high b) Reception of multiple frames **c)** **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **data** gets set to the new value after EOF, **error** is low 4) **Requirement 3, 5, 10 (c)** a) Set **RXD** randomly for 10^6 bit frames, then send 24-bit preamble/SFD followed by a byte of data and	

period in a larger frame (one bit in one byte is low). (e) deliberately wrong input where transmission of a byte ends prematurely (only 6 bits of a byte) (f) attach the receiver to the previously design transmitter and test frames of sizes 1, 256, and 10 randomly sized frames(1-256). (g) same as (f) but with different bit rates (test +- 1%) 11. Testbenches should print a summary of results with PASS or FAIL 12. In Hardware, using the mxtest given, attach the previously designed manchester receiver and generate short frames that will be displayed on the 7-segments as well as RealTerm (after going through a FIFO buffer module). 13. Upload testbenches to Moodle for other groups to test their receivers with. 14. No latches	EOF then set **RXD** randomly for 10^6 bit frames b) Reception of one frame **c)** **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **data** gets set to the new value after EOF, **error** is low 5) **Requirement 6, 10 (d)** a) Set **RXD** high then send 24-bit preamble/SFD followed by a single byte of data with an incorrect bit and EOF then set **RXD** high b) Error detection of a manchester bit **c)** **error** is high once the bit with no transition is detected and stays high until a new frame is detected, **cardet** is high after the first 8 bits of the preamble then low after the error is detected, **data** doesn't get set 6) **Requirement 10 (e)** a) Set **RXD** high then send 24-bit preamble/SFD followed by a 6 bits of a byte of data and EOF then set **RXD** high b) Error detection of a byte **c)** **error** is high once the EOF is seen and stays high until a new frame is detected, **cardet** is high after the first 8 bits of the preamble then low after the error is detected, **data** doesn't get set 7) **Requirement 10 (f)** a) Connect our Manchester Transmitter to our receiver and send one	

	byte b) Connection to our receiver **c) Data** is displayed on RealTerm and on the 7-segment display and should be the byte, **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **error** is low 8) **Requirement 10 (f)** a) Connect our Manchester Transmitter to our receiver and send 256 bytes b) Connection to our receiver **c) Data** is displayed on RealTerm and on the 7-segment display and should be the byte, **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **error** is low 9) **Requirement 10 (f)** a) Connect our Manchester Transmitter to our receiver and send 10 random values of bytes 1-256 b) Connection to our receiver **c) Data** is displayed on RealTerm and on the 7-segment display and should be the byte, **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **error** is low 10) **Requirement 10 (g)** a) Connect our Manchester Transmitter to our receiver and send one byte with plus and minus	

	1% error on the bit rate b) 1% error is still fine for our receiver **c) Data** is displayed on RealTerm and on the 7-segment display and should be the byte, **Cardet** is high after the first 8 bits of the preamble then low after the EOF, **error** is low	
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: 10/31/2017*	

Simulation Output

```
Tested with 1% BIT_RATE Difference between transmitted and received

=================Testing Simulation test 1=================
OK: write is correct on reset
OK: error is correct on reset
OK: data is correct on reset
OK: write is correct on reset
OK: error is correct on reset
OK: data is correct on reset
=================Testing Simulation test 1.5=================
OK: Cardet is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet goes low after if nothing else seen
=================Testing Simulation test 2=================
OK: Cardet is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Write goes high after one byte
```

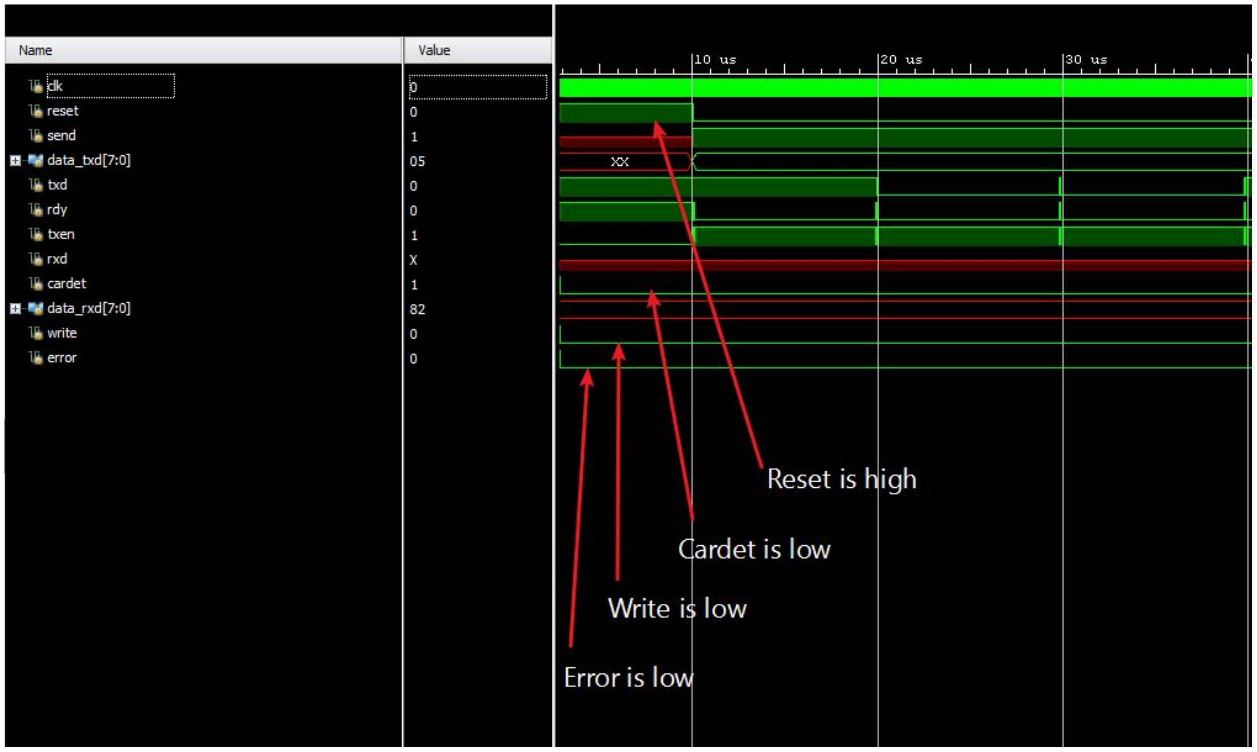
```
OK: Data is as expected 11001100
OK: Write goes low by EOF
OK: Cardet goes low after EOF
=================Testing Simulation test 3=================
OK: Cardet goes is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Write goes high after one byte
OK: Data is as expected 11001100
OK: Write goes high after second byte
OK: Data is as expected 10101010
OK: Write goes high after third byte
OK: Data is as expected 00001111
OK: Write goes low by EOF
OK: Cardet goes low by EOF
=================Testing Simulation test 3.5=================
OK: Cardet goes is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Write goes high after one byte
OK: Data is as expected 11001100
OK: Write goes high after second byte
OK: Data is as expected 10101010
OK: Write goes high after third byte
OK: Data is as expected 00001111
OK: Write goes high after second byte
OK: Data is as expected 10101010
OK: Write goes low by EOF
OK: Cardet goes low by EOF
====================Testing Simulation test 4========================
OK: No preamble was detected during reception of random bits
OK: Cardet goes is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Write goes high after one byte
OK: Data is as expected 11001100
OK: Write goes low by EOF
OK: Cardet goes low after EOF
OK: No preamble was detected during reception of random bits
=================Testing Simulation test 5=================
OK: No preamble was detected during reception of random bits
OK: Cardet goes is low before 8 bit preamble
OK: Cardet goes high after 8 bit preamble
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Error goes high when the bad bit is seen
OK: Cardet has gone low after error
OK: Write goes low by EOF
OK: Cardet goes low by EOF
OK: Error is still high after EOF
OK: No preamble was detected during reception of random bits
=================Testing Simulation test 6==================
OK: Cardet is still high after 16 bit preamble
OK: Cardet is still high after sfd
OK: Error goes high when the EOF is seen
OK: No preamble was detected during reception of random bits


===========SENDING 10_RANDOM_LENGTH BYTE DATA =================
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
```

```
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
```

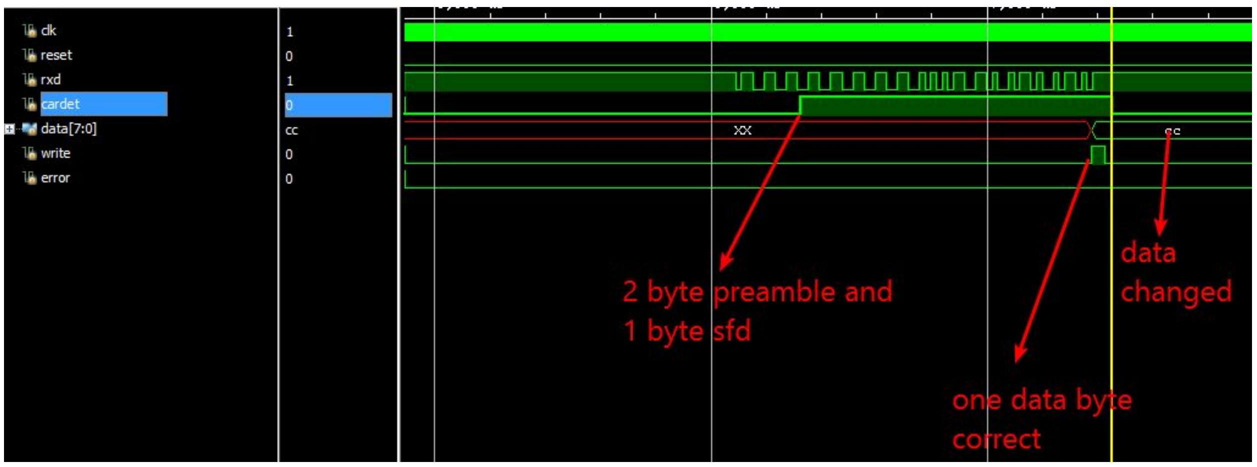
```
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
OK: Cardet is asserted high
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Write signal went high as expected
OK: Data transmitted is data received
OK: Check random number of bytes transmission without error
OK: Check random number of bytes transmission cardet low
```

Waveforms

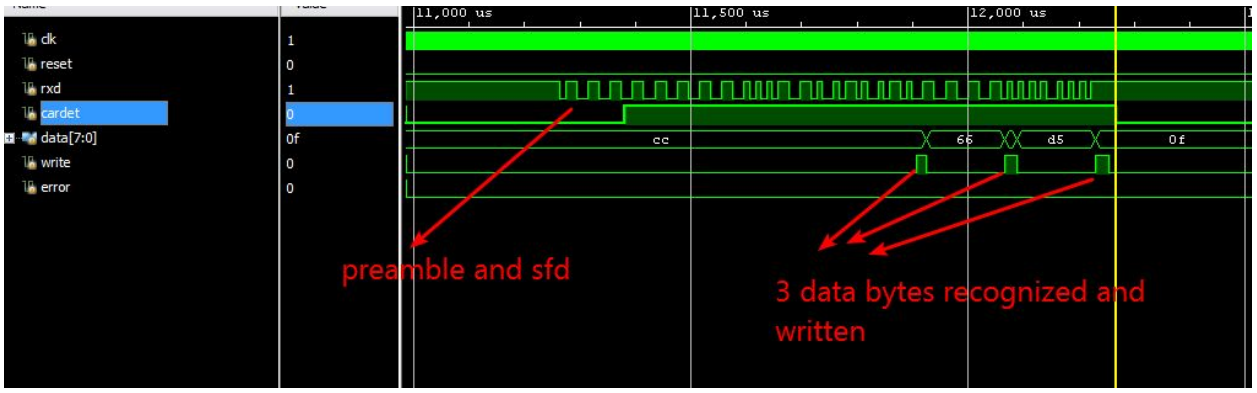
Simulation Test 1 waveform



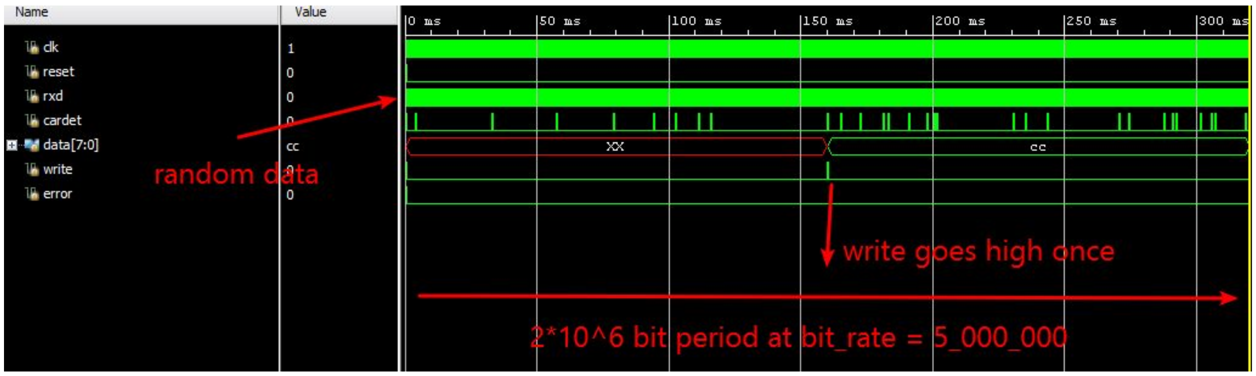
Simulation Test 2 waveform



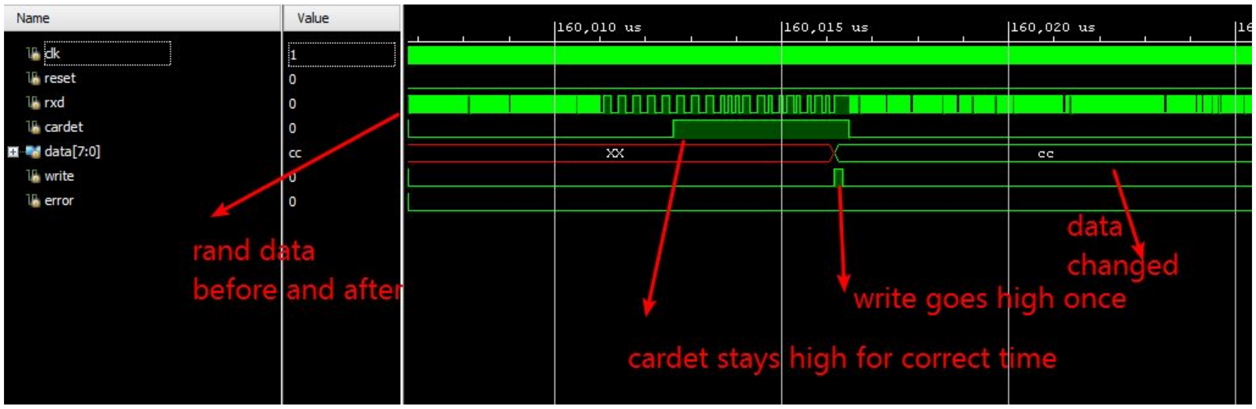
Simulation Test 3 Waveform



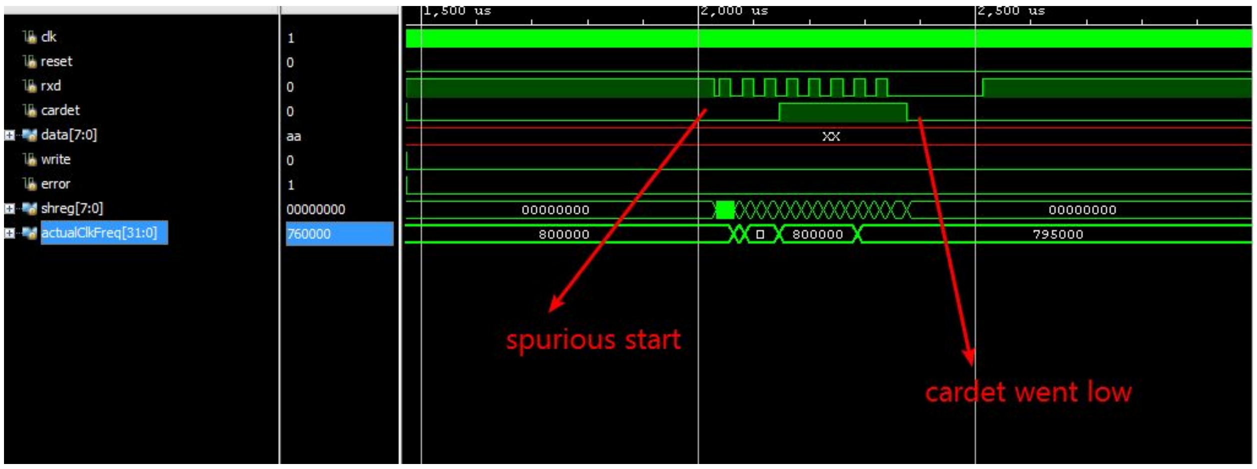
10^6 bit periods before and after signal (**Simulation test 4**)



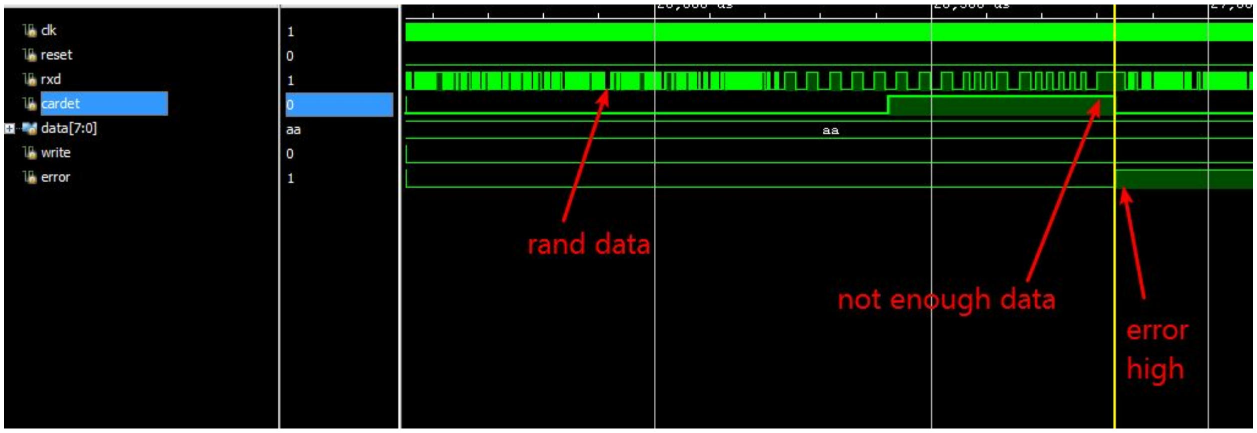
Zoomed in image



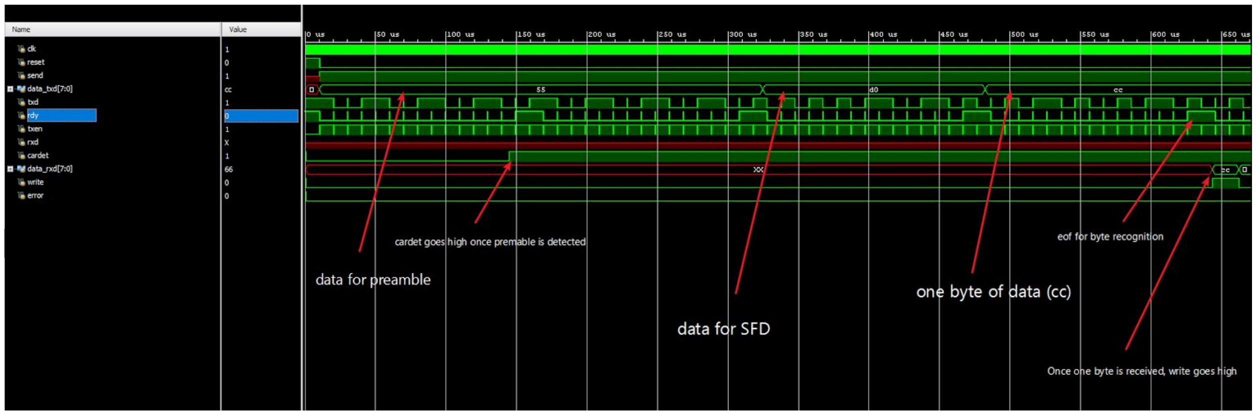
Simulation Test 5 waveform



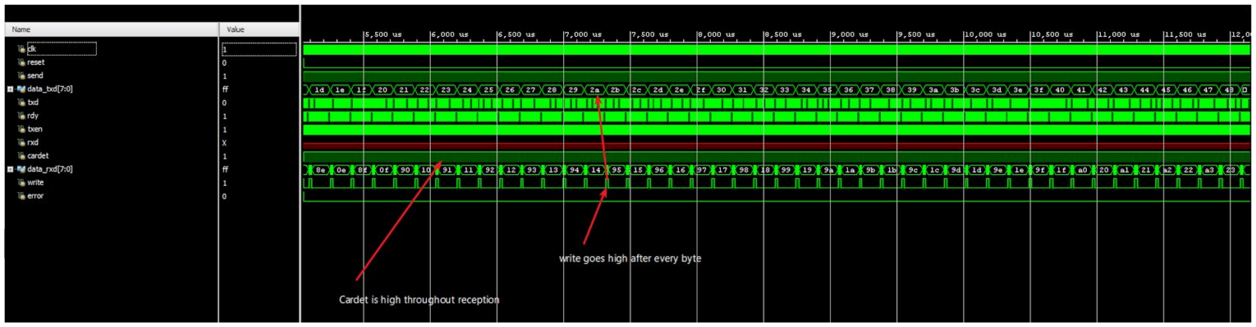
Simulation Test 6 waveform



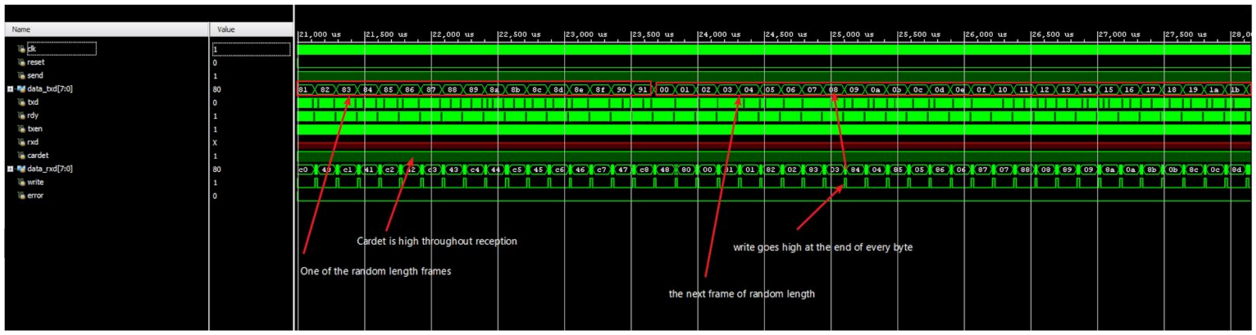
Simulation Test 7 waveform



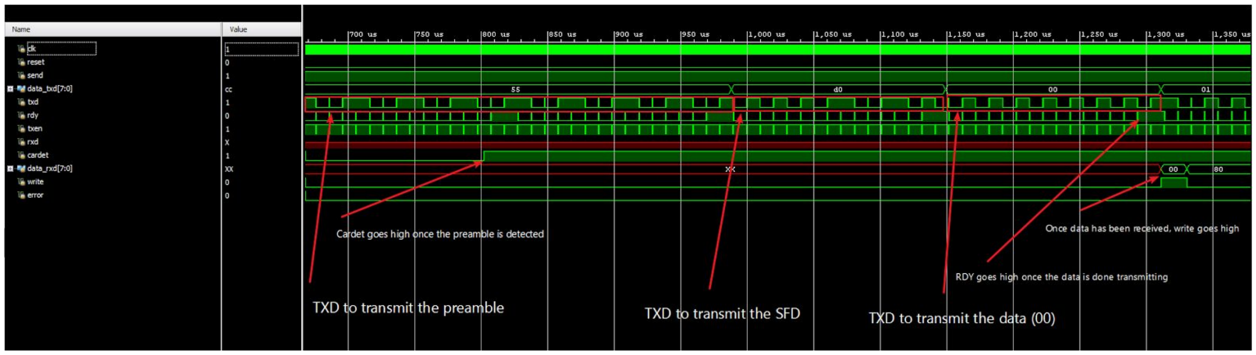
Simulation Test 8 waveform



Simulation Test 9 waveform


Cardet is high throughout reception
One of the random length frames
write goes high at the end of every byte
the next frame of random length

Simulation Test 10 waveform


Cardet goes high once the preamble is detected
Once data has been received, write goes high
RDY goes high once the data is done transmitting
TXD to transmit the preamble
TXD to transmit the SFD
TXD to transmit the data (00)

SIM_OUT Ian & Peter

```
OK: Verify carrier detected (test 1b)
OK: Verify cardet still high (test 1c)
OK: Verify cardet low (test 1c)
OK: Verify 1 byte received (test 1d)
OK: Verify data valid (test 1d)
OK: Verify no errors (test 1e)
OK: Checking single byte transmission (test 1) (7 tests passed)
START: Checking multiple byte transmission (test 2) at time 1088056.
OK: Verify carrier detected (test 2b)
OK: Verify cardet still high (test 1c)
OK: Verify cardet low (test 1c)
OK: Verify 1 byte received (test 1d)
OK: Verify no errors (test 1e)
OK: Checking multiple byte transmission (test 2) (12 tests passed)
START: Send low rxd for bit duration (test 4) at time 3456072.
OK: Verify carrier detected (test 4c)
OK: Error asserted (test 4b)
OK: Cardet deasserted (test 4c)
```

```
OK: Error still asserted (test 4d)
OK: Send low rxd for bit duration (test 4) (12 tests passed)
START: Terminate transmission early (test 5) at time 6512088.
OK: Verify carrier detected (test 5c)
OK: Error asserted (test 5b)
OK: Cardet deasserted (test 5c)
OK: Error still asserted (test 5d)
OK: Verify carrier detected (test 5e)
OK: Verify error cleared (test 5e)
OK: Terminate transmission early (test 5) (12 tests passed)
START: Send preamble with no SFD (test 9) at time 9616104.
OK: Verify carrier detected (test 9a)
OK: Verify carrier detected (test 9a)
OK: Verify carrier lost (test 9a)
OK: Verify carrier detected (test 9a)
OK: Verify carrier lost (test 9a)
*FAIL(Verify no error) Failed test 63 at time 12016136.
  Expected value 0x0, Inspected Value 0x1
FAIL: Send preamble with no SFD (test 9). (1/18 falures)
START: Checking single byte transmission (test 1) at time 12176152.
OK: Verify carrier detected (test 1b)
OK: Verify cardet still high (test 1c)
OK: Verify cardet low (test 1c)
OK: Verify 1 byte received (test 1d)
OK: Verify data valid (test 1d)
OK: Verify no errors (test 1e)
OK: Checking single byte transmission (test 1) (7 tests passed)

Tesbench Complete.
ATTENTION: 1 Error(s) in 71 tests
```

***The error aboves are due to a design decision. After correct preamble and SFD, if rxd is low, our design interprets it as an error, whereas the other design interprets it as not.**