# EE-492-S02-SpTp-Advance Digital Hardware-SP2017

Project #1

Design and Verification of a   
UART – RS232 Asynchronous Serial Communication Interface

Written by: [Jordan D. Ulmer](https://sites.google.com/site/whoisjdu/home)

Written by: [Nathan](https://sites.google.com/site/whoisjdu/home) Genetzky

Instructor: Dr. Robert Fourney

|  |  |
| --- | --- |
| Design Initiated: | **2017-02-26** |
| Date Hardware Demonstrated: | **2017-03-03** |
| Report Submission: | **2017-03-08** |
| Git Repository: | <https://bitbucket.org/adhd_digital_design/ee492> |

## Background:

A Universal Asynchronous Receiver Transmitter (UART), modeled after the specifications detailed by section 9.6, titled “Asynchronous Serial Communication Interface”, of the M68HC11 microcontroller. The original project proposal ([See Appendix A](#_2et92p0)) was intended to allow the UART modules to be designed to meet specifications and requirements rather than simply implementing a given design.

## Verification Summary:

The RS232 modules were verified both within a simulation environment and within an FPGA for various baud rates, data widths, and parities (odd, even and none). This section provides a summary of the tests performed, but the next section shows a detailed look at the hardware test procedure. Table 1, shows the results for both simulation and hardware testing at various Baud Rate and parity configurations (tests performed using Firmware version [5c96059](https://bitbucket.org/adhd_digital_design/ee492) from 2017-03-06).

See Appendix D - Simulation Logfile Results

**Table 1: Standard Baud Rate Simulation and Hardware Verification Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Baud Rate(Hz)** | **Parity** | **Simulation Result (All Assertions Pass)** | **Hardware TX (Sent 0xFF, 0xAA)** | **Hardware RX (Verified with hex\_count\_up.bin)** |
| **115200** | **ODD** | PASS-2017-03-06 | PASS-2017-03-06) | PASS-2017-03-06 |
| **9600** | **ODD** | PASS-2017-03-06 | PASS-2017-03-06 | PASS-2017-03-06 |
| **110** | **ODD** | \*SIM Takes too Long Assertions that were run Passed | PASS-2017-03-06 | PASS-2017-03-06 |
| **115200** | **EVEN** | PASS-2017-03-06 | PASS-2017-03-06 | PASS-2017-03-06 |
| **9600** | **EVEN** | PASS-2017-03-06 | PASS-2017-03-06 | PASS-2017-03-06 |
| **110** | **EVEN** | Did Not Test | PASS-2017-03-06 | PASS-2017-03-06 |
| **115200** | **NONE** | PASS-2017-03-06 | PASS-2017-03-06 | PASS-2017-03-06 |
| **9600** | **NONE** | PASS-2017-03-06 | PASS-2017-03-06 | PASS-2017-03-06 |
| **110** | **NONE** | Did Not Test | PASS-2017-03-06 | PASS-2017-03-06 |

## Standard Hardware Verification Procedure, Results and Analysis:

For hardware verification, the DE2 FPGA development board was connected to a PC that used RealTerm, a serial terminal for Windows, to communicate via RS232. The receive functionality for the RS232 module was tested by sending a standard ascii file from RealTerm to the DUT and visually monitoring the flags and data received on hex displays, for each character. Notably, RealTerm was configured so that there was a 1s delay between each character so there was time to verify the results. The hexadecimal values of the transmitted characters, from the file hex\_count\_up.bin) were: 0xEF, 0xBB, 0xBF, 0x20, 0x01, 0x20, 0x02, 0x20, 0x03, 0x20, 0x04, 0x20, 0x05, 0x20, 0x06, 0x20, 0x07, 0x20, 0x08.

The basic transmit functionality of the RS232 module passed verification. Using ODD Parity in RealTerm and an ODD Parity on the DE2 Dev Board. A data sequence (0xF5, 0xFF, 0x00, 0x01, 0xAA, 0xDE) was correctly transmitted from the firmware on the FPGA; for each data packet transmitted it was we visually verified that NO parity error and NO framing error were detected by RealTerm, Fig. 1. The transmit functionality was further verified by sending a constant value continuously at the end of the manual sequence, Fig.2.



###### Figure 1: RS232 TX Verification Passed without Framing Error and Without



###### Figure 2: RS232 Continuous TX Verification

The basic receive functionality of the RS232 module passed verification. Using odd Parity in RealTerm and the firmware on the DE2 development board. The firmware correctly received the data without a parity error and without a framing error. The screenshot of RealTerm, picture of the development board and a summary are shown, Fig.3.



###### Figure 3: RS232 RX Verification

The parity error detection in the receive module of the RS232 module passed verification. This was done by using an even parity in RealTerm and an odd parity in the firmware. The parity LED (yellow box) indicated that the parity error was correctly identified by the firmware on the FPGA, Fig.4.



###### Figure 4: RS232 RX Parity Error Verification

The framing error detection of the receive module passed verification. This was done by using even parity in RealTerm and no parity in the firmware; this means that RealTerm will be sending a parity bit as the 8th bit, rather than the STOP BIT that the firmware expects. When the data packet was sent from RealTerm, a framing error was correctly identified by the DUT as shown by the framing error LED (yellow box), Fig.5.



###### Figure 5: RS232 RX Framing Error Verification

## Baud Rate Mismatch Hardware Verification Results and Analysis:

The maximum baud rate mismatch for a 50MHz oscillator was calculated using [“WormFood's AVR Baud Rate Calculator”](http://wormfood.net/avrbaudcalc.php?bitrate=9600&clock=50&databits=10&hidecolors=1), Table 6.

The actual maximum and minimum mismatch for three baud rates were tested by sending the file described earlier (hex\_count\_up.bin). Notably, the RS232 design consistently performed within specifications set forth in Table 2 for baud rates of 115200 Hz and 110 Hz, Table 3; However, at 9600 Baud the RS232 design was 53 Hz out of spec. for the minimum baud rate(See Tables below). Detailed notes were taken when performing the baud rate mismatch verification, Fig.8.

###### Table 2: Allowable RX Bit Rate Error as a percent of the Baud Rate as specified by RS232 specification. Credit: [“WormFood's AVR Baud Rate Calculator”](http://wormfood.net/avrbaudcalc.php?bitrate=9600&clock=50&databits=10&hidecolors=1)



###### 

###### Table 3: RS232 Baud Rate Mismatch Verification Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Baud Rate(Hz)** | **Parity** | **Expected Maximum Baud Rate(Hz)** | **Actual Maximum Baud Rate(Hz)**  **Hardware RX**  **(Verified with hex\_count\_up.bin)** | **Expected Minimum Baud Rate(Hz)** | **Actual Minimum Baud Rate(Hz)**  **Hardware RX**  **(Verified with hex\_count\_up.bin)** |
| **115200** | **ODD** | 119969.28 | 121000 | 110373.12 | 100000 |
| **9600** | **ODD** | 9997.44 | 10000 | 9197.76 | 9250  (53Hz from spec)  (96% Of 9600)  (Expected 95.81% Of 9600) |
| **110** | **ODD** | 114.554 | 130 | 105.391 | 100 |

###### Table 4: RS232 Baud Rate Mismatch Verification Notes

|  |  |  |
| --- | --- | --- |
| **Baud Rate(Hz)** | **Parity** | **HW Notes**  **JDU@GIT:(5c96059)** |
| **115200** | **ODD** | RX fail @ 122000  Rx Noise Flag on all TX from RealTerm @ 115200  NO Rx Noise flag, when 0xFF sent from RealTerm @ 110373  Rx Noise flag, when 0xFF sent from RealTerm @ 110000  Rx Noise flag, when 0xAA sent from RealTerm @ 100000  NO Rx Noise flag, when 0xEC sent from RealTerm @ 100000  NO Flags but Rx=0xFF, when 0xAA sent from RealTerm @ 90000 |
| **9600** | **ODD** | All Flags wrong Rx, sending 0xAA four-times with no delay @ 10500  Noise Flag, Rx=0xAA , sending 0xAA once @ 10500  Noise Flag, Rx=0xAA sending 0xAA four-times with no delay @ 10000  Rx Fail @ 9197  Noise Flag, Rx Pass sending ascii @ 9250  Noise Flag, Rx=0xAA , sending 0xAA four-times with no delay @ 9250  Noise Flag, Parity Flag, Rx Wrong, sending 0xAA four-times with no delay @ 9225 |
| **110** | **ODD** | Noise Flag, Rx=0xAA , sending 0xAA four-times with no delay @ 114  Noise Flag, Rx=0xAA , sending 0xAA four-times with no delay @ 120  Noise Flag, Rx=0xAA , sending 0xAA four-times with no delay @ 130  Noise Flag for some Rx, all char correct, sending hex\_count\_up.bin @ 130  Rx Fail @ 140  Noise Flag, Rx=0xAA , sending 0xAA four-times with no delay @ 100  Noise Flag for some Rx, all char correct, sending hex\_count\_up.bin @ 100  Rx Fail @ 90 |

## Simulation Verification Results and Analysis:

### Start Bit detection and Noise Flag

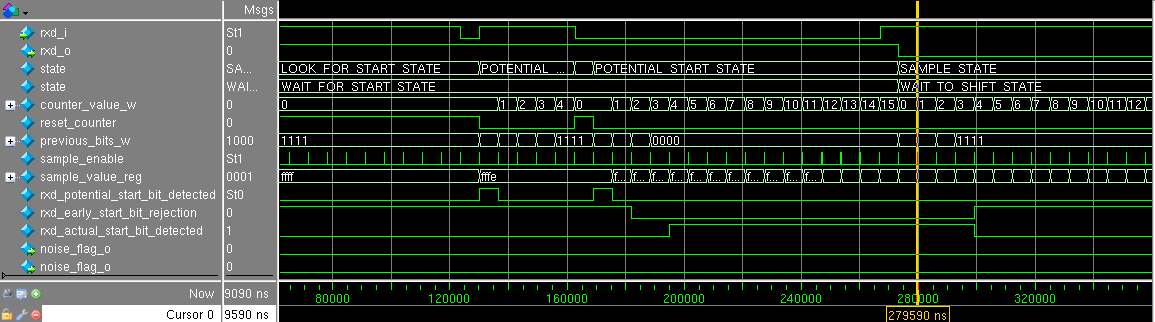
In order to verify that our design meets the specifications detailed in [section 9 for the M68HC11 microcontroller](https://drive.google.com/open?id=0B-DYXcrmLV3cWHpkTGJycXpkUjg), the specific noise cases presented were replicated in an SystemVerilog Testbench and and the results presented in this section. The referenced section in the manual described each of these noise cases in great detail, but they will only be briefly described here. All of these cases described the value of the RX pin and the expected results, such as whether the start bit was detect or if a noise condition occurred. First, the ideal case is presented, Fig.9; the START BIT is present on the RX pin for 16 sample periods (1 baud period). As expected the start bit is detected and no noise flag is set.

###### sb0.png

###### Figure 9: Start Bit - Ideal Case

Next, the more interesting cases are considered. In some scenarios the start bit will not be detected, but in others it will be detected with a noise flag. Noise Case One shows a scenario where noise causing a false START BIT is completely ignored when it is sufficiently far from the actual start bit, Fig.10. Alternatively, Noise Case two shows that if the noise is close to the actual start bit then the start bit will be detected early, Fig.11; note that the noise flag is properly set due to this noise condition.

This noise flag is latched in the rx\_in module, which uses the data\_recovery for start bit detection; this flag will be available to the user of the RS232 module for the byte received after this noise case. This is important because it means that each of the following bits will not be sampled in the middle (samples 7,8,9) of the bit; however, the shifted sampled will still occur far enough from the data bits signal transitions that the bits will be properly received.

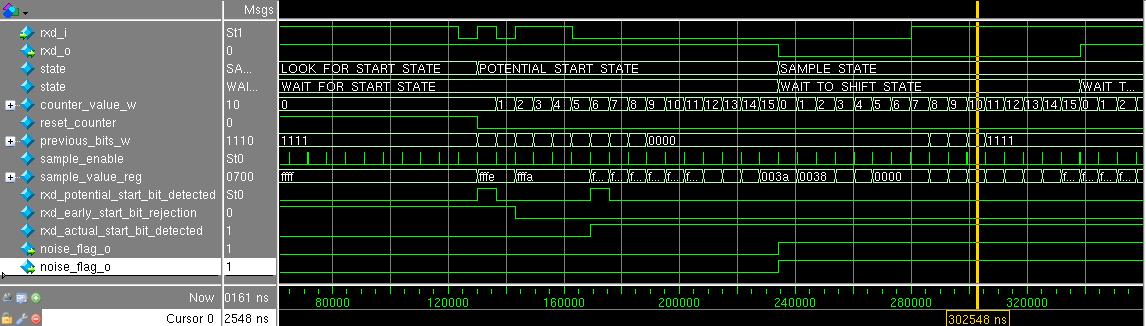


###### Figure 10: Start Bit - Noise Case One

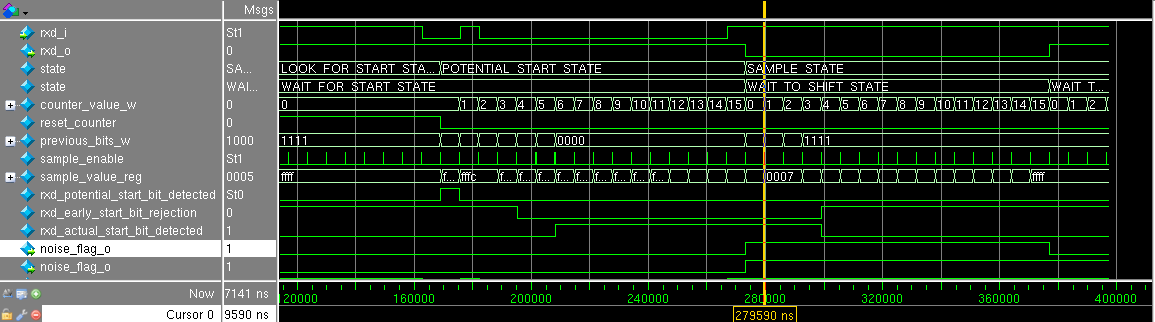
###### sb2(1).png

###### Figure 11: Start Bit - Noise Case Two

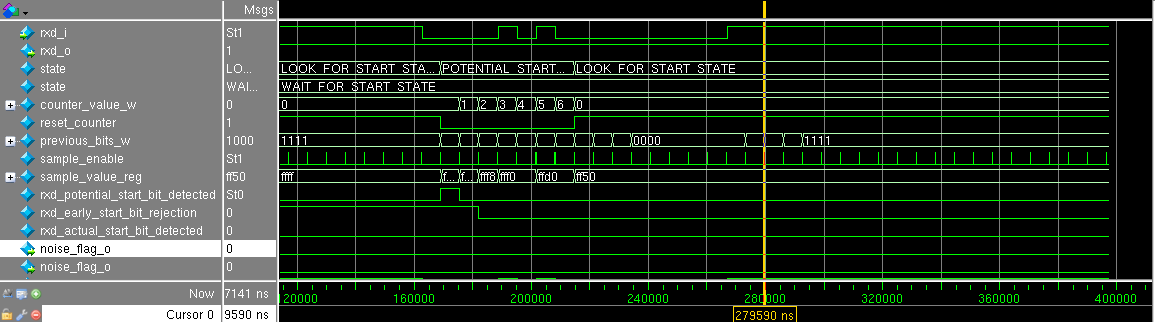
Noise Case Three demonstrates how the data\_recovery module can handle extreme amounts of noise and still recover the intended data. Due to the noise the perceived start bit is offset from the actual start bit and the noise flag is set at the end of the perceived START BIT, Fig.12. Noise Case Four shows an example of noise that has no effect on the perceived start bit but will still set the noise flag, Fig.13. The Noise Case Five shows a scenario where the data\_recovery module fails to detect the START BIT, Fig.14; this will lead to a framing error later, but that is not shown. Finally, Noise Case Six shows how special logic for the START BIT can cause it to be detected even when two out of three voting fails; obviously, the noise flag is set in this case.



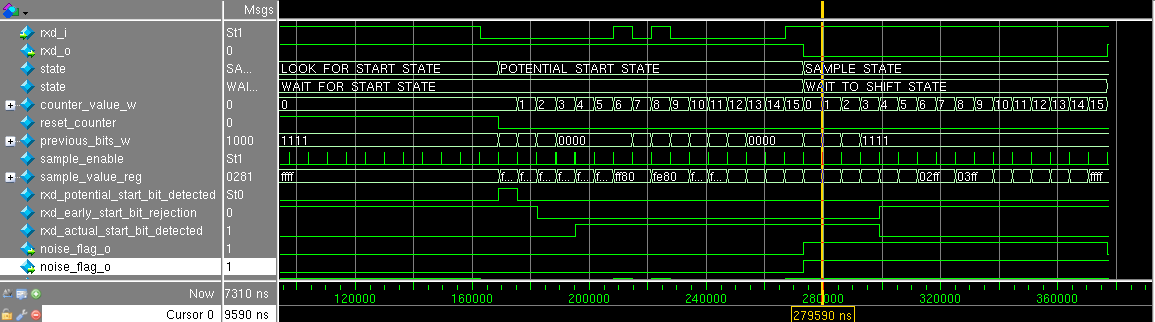
###### Figure 12: Start Bit - Noise Case Three



###### Figure 13: Start Bit - Noise Case Four



###### Figure 14: Start Bit - Noise Case Five



###### Figure 15: Start Bit - Noise Case Six

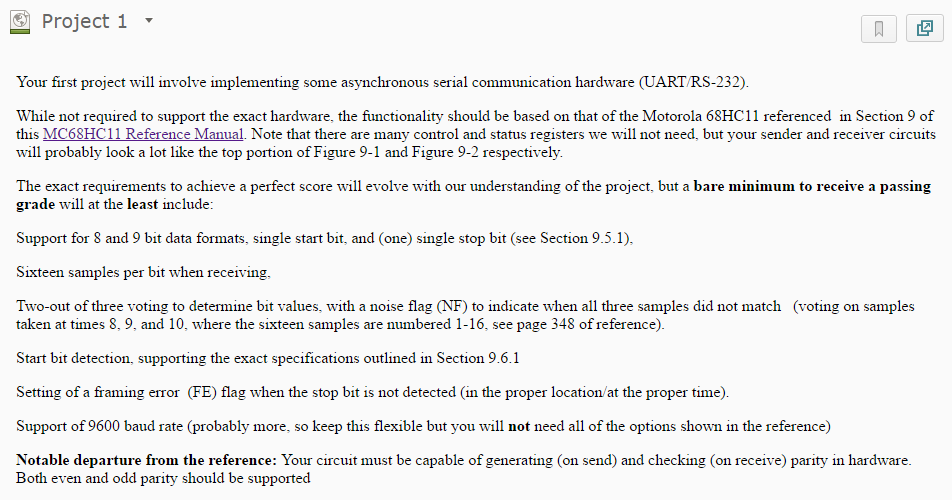
## Conclusion:

The requirements set forth for this project have been rigorously tested and verified. In this project, the designers identified two primary lessons learned :

1. Optimize design FSMs for speed by eliminating latches, this can be accomplished through sequential combinational logic.
2. The use of the “assert” keyword in SystemVerilog testbenches provides ease of code readability and is prefered to “if” statements where appropriate.

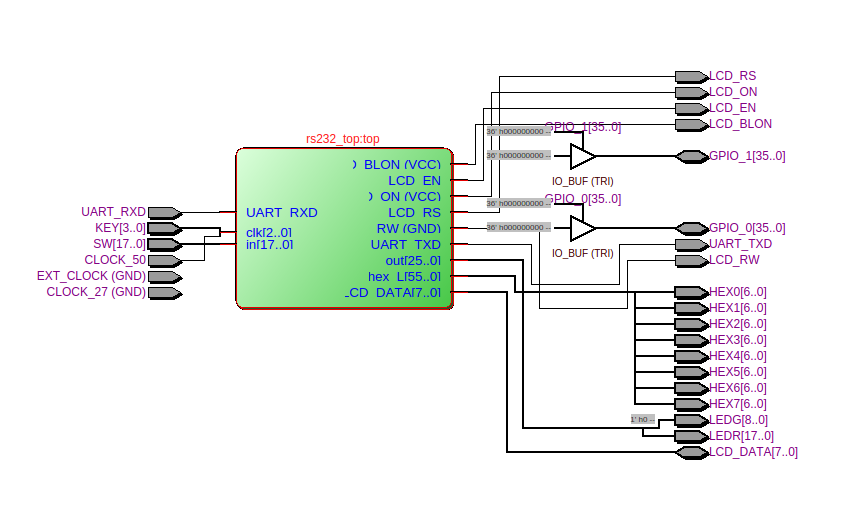
# Appendices:

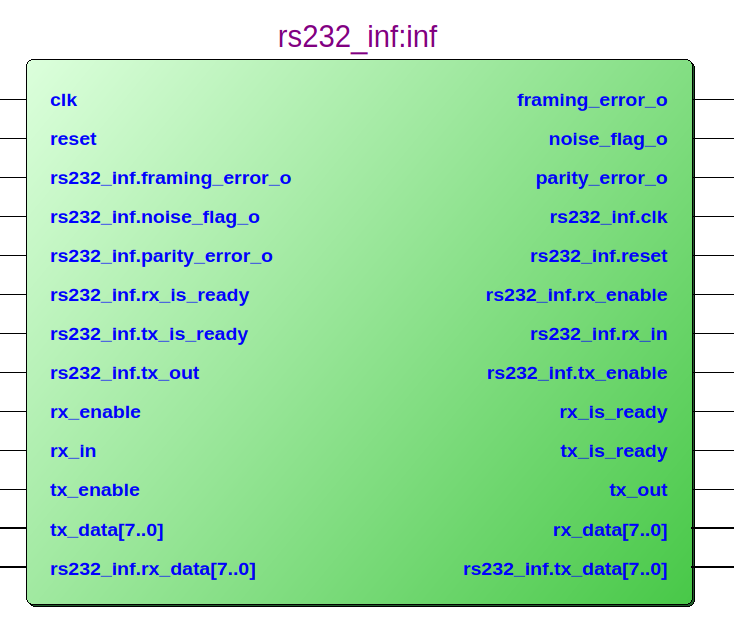
## A – Original Assignment:

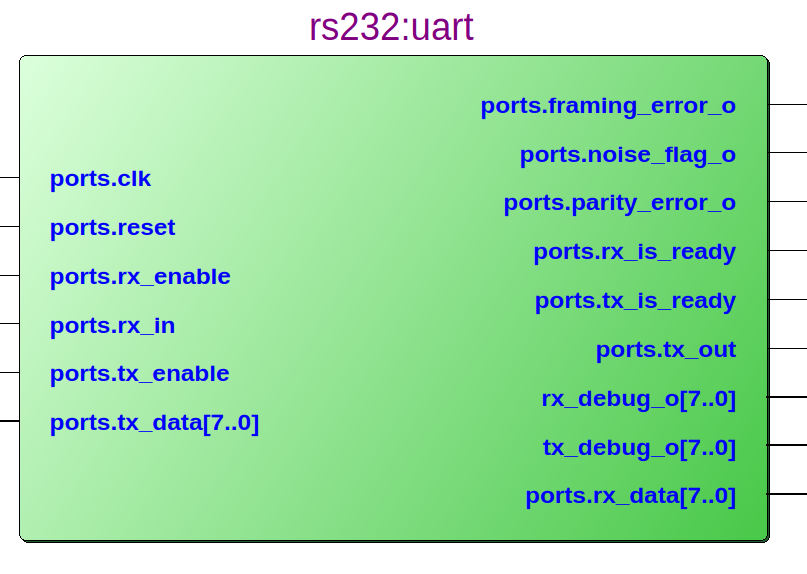


###### Figure 16: Original Project 1 Assignment

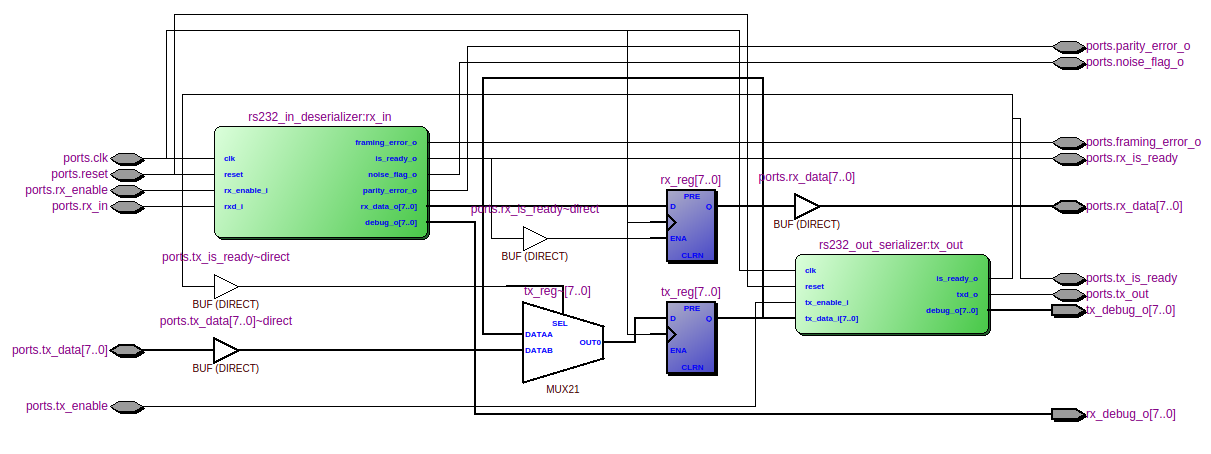
## B – rs232\_JUNG RTL Breakdown:

 **Figure 17: rs232\_top module, shows DE2 I/O, LCD was setup in preparation for FIFOS, but was not used**

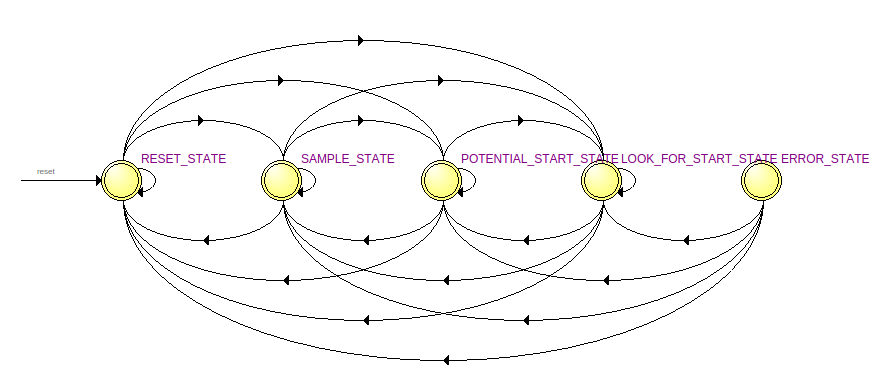
  
**Figure 18: rs232\_inf Modularized Interface**

 **Figure 19: rs232:uart contains TX and RX logic**

Notably, the rs232\_JUNG design, latches the rx\_data\_o and tx\_data\_i inside the rs232:uart module. FIFOs

**Figure 20: Inside rs232:uart - TX and RX logic and rx\_reg , tx\_reg**

Notably, the rs232\_JUNG design uses proper FSM conventions, providing a standard RESET state and default ERROR state , the SystemVerilog “typedef enum” keyword was used to have the compiler optimize the state values.

 **Figure 21: Finite State Machine for rs232:uart|rs232\_in\_deserializer:rx\_in|data\_recovery:sampler|state**

## C - Future Work:

The designers have identified several actions for future development and usage of this design:

1. Modify rs232\_JUNG to communicate with FTDI chip through SCI.
   1. SCI to FTDI
   2. SCI to JTAG wrapper module (To talk with HIPE)
2. Modify the module architecture rs232\_JUNG to work as a more [generalized SCI](http://ume.gatech.edu/mechatronics_course/SCI%20F13.pptx) (See Figure below).
3. Wrap rx\_in\_deserializer and rx\_out\_serializer with a FIFOs and controller.

###### Figure 17: Example Block Diagram of a Generalized SCI Architecture (Source: gatech.edu)

## D – Simulation Logfile Results:

A rigorous assertion based simulation approach was performed to ensure the rs232\_JUNG design performs as specified.

###### Table 5: Modelsim Transcript for 9600 Baud ODD Parity

|  |
| --- |
| File: /home/caeuser/workspace/jdulmer/ee492\_team/projects/RS232/modelsim/transcript.rs232\_p1  001: # vsim -do {do ./default.do.tcl} -c ././/work.simple\_top  002: # Loading sv\_std.std  003: # Loading ././/work.simple\_top  004: # Loading ././/work.rs232\_inf  005: # Loading ././/work.rs232  006: # Loading ././/work.rs232\_out\_serializer  007: # Loading ././/work.shift\_reg\_n  008: # Loading ././/work.pulse\_generator  009: # Loading ././/work.clk\_divider  010: # Loading ././/work.rs232\_in\_deserializer  011: # Loading ././/work.counter\_wreset  012: # Loading ././/work.data\_recovery  013: # Loading ././/work.parity  014: # do ./default.do.tcl  015: # top::initial  016: # 1488775028433895605  017: # rs232\_reset();  018: # [ 0] 9600hz =BAUD\_RATE  019: # [ 0] 9600hz =CHECK\_BAUD\_RATE  020: # [ 0] 104160ns =T\_BAUD  021: # [ 0] 20ns =T\_CLOCK  022: # test\_start\_bit();  023: #  024: # \*\* Info: PASS: Correctly verified noise flag NOT SET for mock\_start(0)  025: # Time: 293021 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_0\_fig\_9\_10 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 372  026: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(0)  027: # Time: 293021 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_0\_fig\_9\_10 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 373  028: #  029: #  030: # \*\* Info: PASS: Correctly verified noise flag NOT SET for mock\_start(1)  031: # Time: 1835921 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_1\_fig\_9\_11 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 425  032: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(1)  033: # Time: 1835921 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_1\_fig\_9\_11 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 426  034: #  035: #  036: # \*\* Info: PASS: Correctly verified noise flag was SET for mock\_start(2)  037: # Time: 3378821 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_2\_fig\_9\_12 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 483  038: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(2)  039: # Time: 3378821 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_2\_fig\_9\_12 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 484  040: #  041: #  042: # \*\* Info: PASS: Correctly verified noise flag was SET for mock\_start(3)  043: # Time: 4934741 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_3\_fig\_9\_13 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 538  044: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(3)  045: # Time: 4934741 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_3\_fig\_9\_13 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 539  046: #  047: #  048: # \*\* Info: PASS: Correctly verified noise flag was SET for mock\_start(4)  049: # Time: 6477641 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_4\_fig\_9\_14 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 574  050: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(4)  051: # Time: 6477641 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_4\_fig\_9\_14 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 575  052: #  053: #  054: # \*\* Info: PASS: Correctly verified noise flag was NOT SET for mock\_start(5)  055: # Time: 8020541 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_5\_fig\_9\_15 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 624  056: # \*\* Info: PASS: Correctly did NOT receive START\_BIT for mock\_start(5)  057: # Time: 8020541 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_5\_fig\_9\_15 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 625  058: #  059: #  060: # \*\* Info: PASS: Correctly verified noise flag for mock\_start(6)  061: # Time: 9543610 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_6\_fig\_9\_16 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 667  062: # \*\* Info: PASS: Correctly received START\_BIT for mock\_start(6)  063: # Time: 9543610 ns Scope: simple\_top.rx\_mock\_start\_bit.noise\_6\_fig\_9\_16 File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 668  064: #  065: # try\_special();  066: # \*\* Info: PASS: Correctly received (01)  067: # Time: 12152391 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  068: #  069: #  070: # \*\* Info: PASS: Correctly received (02)  071: # Time: 13511231 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  072: #  073: #  074: # \*\* Info: PASS: Correctly received (80)  075: # Time: 14870071 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  076: #  077: #  078: # \*\* Info: PASS: Correctly received (c0)  079: # Time: 16228911 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  080: #  081: #  082: # \*\* Info: PASS: Correctly received (aa)  083: # Time: 17587751 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  084: #  085: #  086: # \*\* Info: PASS: Correctly received (af)  087: # Time: 18946591 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  088: #  089: #  090: # \*\* Info: PASS: Correctly received (a0)  091: # Time: 20305431 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  092: #  093: #  094: # try\_random(10);  095: # \*\* Info: PASS: Correctly received (4e)  096: # Time: 21664271 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  097: #  098: #  099: # \*\* Info: PASS: Correctly received (4c)  100: # Time: 23023111 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  101: #  102: #  103: # \*\* Info: PASS: Correctly received (60)  104: # Time: 24381951 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  105: #  106: #  107: # \*\* Info: PASS: Correctly received (44)  108: # Time: 25740791 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  109: #  110: #  111: # \*\* Info: PASS: Correctly received (26)  112: # Time: 27099631 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  113: #  114: #  115: # \*\* Info: PASS: Correctly received (51)  116: # Time: 28458471 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  117: #  118: #  119: # \*\* Info: PASS: Correctly received (9d)  120: # Time: 29817311 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  121: #  122: #  123: # \*\* Info: PASS: Correctly received (0b)  124: # Time: 31176151 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  125: #  126: #  127: # \*\* Info: PASS: Correctly received (3e)  128: # Time: 32534991 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  129: #  130: #  131: # \*\* Info: PASS: Correctly received (21)  132: # Time: 33893831 ns Scope: simple\_top.wait\_for\_both\_ready File: ../../../firmware/rs232//src\_tb/simple\_top.sv Line: 782  133: #  134: #  135: # top::final  136: # 1488775083115910175  137: |

## E – [TOC] – Table of Contents:

### Table of Contents:

[**EE-492-S02-SpTp-Advance Digital Hardware-SP2017**](#_wfz666irym1n) **1**

[Background:](#_30j0zll) 2

[Verification Summary:](#_g5bsvm99g7jj) 2

[Standard Hardware Verification Procedure, Results and Analysis:](#_qbwyjh1mz25a) 2

[Figure 1: RS232 TX Verification Passed without Framing Error and Without](#_qmsqbv1119b5) 3

[Figure 2: RS232 Continuous TX Verification](#_1h51xszdl2j5) 4

[Figure 3: RS232 RX Verification](#_4p0glx8hq1dw) 5

[Figure 4: RS232 RX Parity Error Verification](#_yqnlw8aup6lz) 6

[Figure 5: RS232 RX Framing Error Verification](#_b61k6fsv2e1p) 7

[Baud Rate Mismatch Hardware Verification Results and Analysis:](#_8ga57qaosbu) 7

[Table 2: Allowable RX Bit Rate Error as a percent of the Baud Rate as specified by RS232 specification.  
Credit: “WormFood's AVR Baud Rate Calculator”](#_tfyjnc6fld5j) 8

[Table 3: RS232 Baud Rate Mismatch Verification Summary](#_keobx2ghkb6t) 8

[Table 4: RS232 Baud Rate Mismatch Verification Notes](#_hkntkt9txyii) 8

[Simulation Verification Results and Analysis:](#_sil2xxj9d48j) 9

[Start Bit detection and Noise Flag](#_5e3mbgm60chn) 9

[Figure 9: Start Bit - Ideal Case](#_unvk95wun5om) 9

[Figure 10: Start Bit - Noise Case One](#_g6o9dtuhs80x) 10

[Figure 11: Start Bit - Noise Case Two](#_s3hkdj7fjel5) 10

[Figure 12: Start Bit - Noise Case Three](#_g7bsc01f0inm) 10

[Figure 13: Start Bit - Noise Case Four](#_qaygnmsuu34h) 11

[Figure 14: Start Bit - Noise Case Five](#_s9ubf9g2pkmh) 11

[Figure 15: Start Bit - Noise Case Six](#_u78j5e3p849x) 11

[Conclusion:](#_g1kwnde32gpg) 11

[**Appendices:**](#_3znysh7) **12**

[A – Original Assignment:](#_2et92p0) 12

[Figure 16: Original Project 1 Assignment](#_3dy6vkm) 12

[B – rs232\_JUNG RTL Breakdown:](#_6pqi1ngt5duq) 13

[C - Future Work:](#_2we6yp6ttct) 16

[Figure 17: Example Block Diagram of a Generalized SCI Architecture   
(Source: gatech.edu)](#_1jf8if21wl3s) 16

[D – Simulation Logfile Results:](#_nrxcw1ive4dr) 17

[Table 5: Modelsim Transcript for 9600 Baud ODD Parity](#_le9ib1272xet) 17

[E – [TOC] – Table of Contents:](#_1t3h5sf) 21

[Table of Contents:](#_2s8eyo1) 21